Graph Engineering in the Era of LLM Agents: From Individual Intelligence to System Intelligence

Yuyuan Feng*, **Zhishang Xiang***, **Chaobin Yang***, **Qichao Ma***, **Zerui Chen**, **Yujing Zhang**, **Ke Huang**, **Chuanjie Wu**, **Zhaoxu Liu**, **Yili Wang**, **Xin He**, **Jiapu Wang**, **Zijin Hong**, **Hao Chen**, **Yuanchen Bei**, **Kun Wang**, **Shengyuan Chen**, **Ningyu Zhang**, **Enyan Dai**, **Linhao Luo**, **Qingyi Pan**, **Qi Wang**, **Wenqi Fan**, **Guangjing Wang**, **Na Zou**, **Yangqiu Song**, **Xin Wang**, **Zechao Li**, **Xia Hu**, **Qing Li**, **Xiao Huang**, **Zhihong Zhang**†, **Jinsong Su**†, **Qinggang Zhang**†,‡, **Yi Chang**†

*Equal Contribution, †Corresponding Authors, ‡Project Leader

Abstract

Large language models (LLMs) have rapidly evolved from language generation models into autonomous agents capable of solving increasingly complex and long-horizon tasks. This evolution has been accompanied by a series of emerging engineering paradigms, including Prompt Engineering for eliciting model capabilities, Context Engineering for managing information access, Harness Engineering for organizing external tools and resources, and Loop Engineering for enabling continual reflection and self-improvement. However, as real-world tasks grow in complexity, a fundamental limitation of individual intelligence emerges: many tasks inherently require heterogeneous expertise, interdependent subtasks, parallel execution, independent verification, and persistent state, and these requirements exceed the organizational capacity of any single agent. Simply augmenting an individual agent's capabilities or context cannot resolve this architectural mismatch. Instead, intelligence must be distributed across multiple specialized agents and organized at the system level. We refer to this capability as System Intelligence: the ability of an agent system to organize and coordinate multiple intelligent components into a coherent, adaptive whole that pursues a shared objective. Achieving System Intelligence, however, demands more than merely increasing the number of agents; it requires explicit structures for organizing work, coordinating heterogeneous agents, and maintaining evolving execution states. In this survey, we introduce **Graph Engineering**, an emerging paradigm for building next-generation agent systems. Unlike previous paradigms that primarily optimize individual interactions or agent-level behaviors, Graph Engineering focuses on constructing explicit, dynamic, and evolving graph structures that represent tasks, agents, and system states. Such graph-based abstractions provide a unified foundation for organizing complex objectives, orchestrating heterogeneous agents, modeling system dynamics, and enabling scalable agent evolution. In this paper, we systematically review the principles, methodologies, and applications of Graph Engineering in the era of LLM agents. All the related resources, including research papers, open-source data, and projects, are collected for the community at https://github.com/DEEP-JLU/Awesome-Graph-Engineering.

Email: qinggangzhang@jlu.edu.cn

Contents

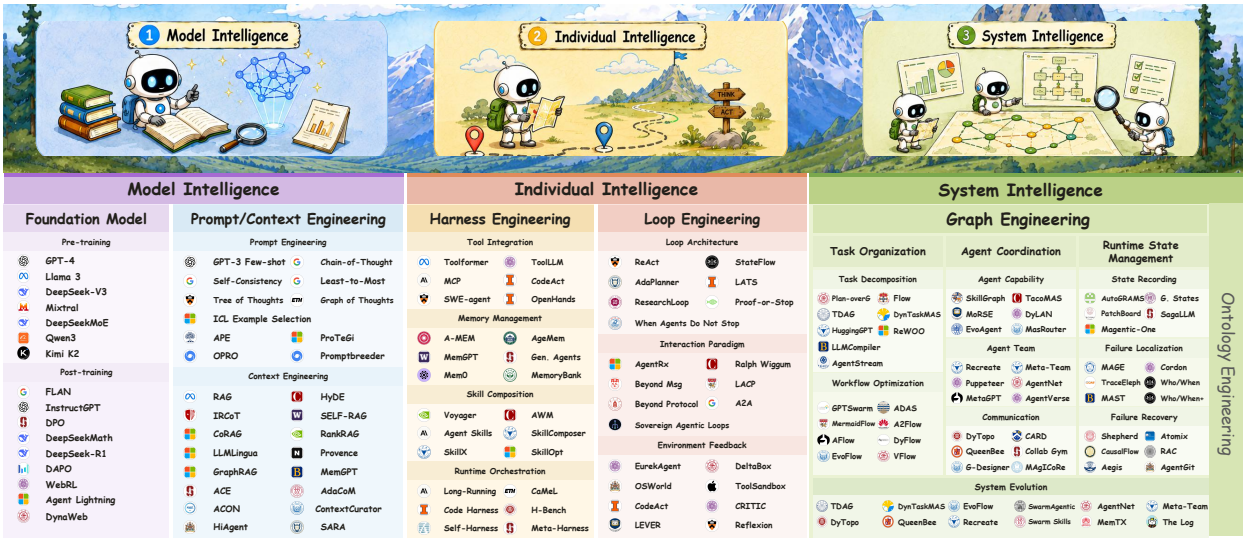

Figure 1 : Overview From Model Intelligence to System Intelligence. Prompt and Context Engineering elicit and condition the access of foundation models to realize model intelligence. Harness and Loop Engineering extend and orchestrate agentic capabilities to enable Individual Intelligence. Graph Engineering builds on the foundation of individual agents through Task Organization, Agent Coordination, and Runtime State Management, empowering System Intelligence.

1 Introduction

Large language models (LLMs) have rapidly evolved into a foundational component of modern intelligent systems, driven by substantial advances in language understanding, reasoning, generation, and decision making [26, 48, 57, 120, 334, 336]. This progress has largely followed two complementary directions: strengthening the capabilities encoded in model parameters during training [52, 221, 267] and improving how these capabilities are activated and utilized at inference time [9, 23, 290, 371]. Specifically, early research primarily focused on the former, using large-scale pre-training and post-training to expand and refine the knowledge and reasoning capabilities of individual models [26, 57, 221, 370]. More recently, increasing attention has shifted toward inference-time engineering, where `Prompt Engineering` and `Context Engineering` serve as complementary approaches for shaping model behavior. `Prompt Engineering` [23, 355, 371] structures task descriptions, instructions, and constraints to guide model reasoning, whereas `Context Engineering` [70, 176, 214] determines and organizes the task-relevant information, external knowledge and intermediate results available to the model during inference. Together, these techniques constitute the primary mechanisms for developing and eliciting **Model Intelligence**, which characterizes the ability of an individual model to leverage its knowledge and reasoning capabilities to solve tasks within a given context.

Despite recent advances in Model Intelligence, its scope remains bounded by what an individual model can access, maintain, and accomplish within a standalone inference process. Many real-world tasks, however, require access to external knowledge and tools, interaction with dynamic environments, and iterative adaptation over extended execution horizons [176, 307, 356]. These requirements have motivated a paradigm shift from developing more capable language models toward constructing autonomous systems around these models, leading to the emergence of LLM-based agents. Conceptually, such an agent can be characterized as:

$$\text{Agent} = \text{Loop}(\text{LLM} + \text{Harness}).$$

Here, `Harness Engineering` extends the capability boundary of the model by connecting it to heterogeneous resources and functional components, including external knowledge [13, 460], tools [287, 290], memory [405, 414], and skills [344, 462]. `Loop Engineering` further organizes these capabilities into a persistent execution process through iterative cycles of planning, action, observation, verification, and adaptation [121, 386]. In this sense, `Harness Engineering` determines what capabilities and resources are available to the agent and how these capabilities are orchestrated during execution, whereas `Loop Engineering` defines how the agent continuously interacts with the environment and adapts its behavior in response to evolving task states and external feedback. Together, they transform an LLM from a response-generating model into a goal-directed autonomous entity capable of sustained

interaction with environments, giving rise to what we term **Individual Intelligence**, the ability of an individual agent to extend model-level reasoning into persistent, goal-directed execution through resource orchestration and iterative interaction with the environment.

However, as real-world tasks grow in complexity, a fundamental limitation of individual intelligence emerges: many tasks inherently require heterogeneous expertise, interdependent subtasks, parallel execution, independent verification, and persistent state, and these requirements exceed the organizational capacity of any single agent [64, 88, 93, 155, 288, 415]. For example, real-world tasks, like scientific discovery and software engineering, often require specialized reasoning to be performed concurrently, intermediate results to be exchanged and validated, and execution states to be maintained across long-running processes. When such tasks are executed within a single agent loop, these heterogeneous processes are forced into a common context and a centralized execution trajectory. This creates several structural bottlenecks: task-relevant information competes for contextual capacity, dependent operations are mediated through a predominantly sequential control process, and the states of different tasks and agents are forced into a single shared context, making it impossible to isolate concurrent work, synchronize on shared results, or recover partial progress independently.

As tasks become increasingly heterogeneous, interdependent, and long-horizon, simply augmenting an individual agent's capabilities or context cannot resolve this problem [110, 214, 288]. Instead, intelligence must be distributed across multiple specialized agents and organized at the system level [159, 284]. We refer to this capability as **System Intelligence**, as shown in Fig. 4: the ability of an intelligent system to decompose and organize complex objectives, allocate responsibilities across heterogeneous computational agents, coordinate their interdependent execution, and maintain system-level state throughout the task lifecycle. Importantly, `System Intelligence` is not equivalent to simply increasing the number of agents. A multi-agent system may contain multiple capable agents while still lacking effective work organization, clear responsibility boundaries, coordination mechanisms, or consistent state management [30, 317]. Moving from `Individual Intelligence` to `System Intelligence` therefore requires a shift from agent-centric execution to system-level organization, where the central challenge is no longer how to replicate or specialize agents, but how heterogeneous components can be organized and coordinated to operate as a coherent system.

To this end, we introduce **Graph Engineering**, a novel engineering paradigm in which graph structures are used to organize and control task execution, agent coordination, and runtime state evolution for system-level intelligence. From a system perspective, as shown in Fig. 2, `Graph Engineering` addresses three fundamental organizational problems. (i) Task Organization determines how a global objective is decomposed into executable units and how dependencies, ordering, concurrency, and verification constraints among them are represented. (ii) Agent Coordination determines how these units of work are mapped onto heterogeneous agents and computational components, and how their communication, delegation, synchronization, and result integration are structured. (iii) Runtime State Management determines how the evolving state of execution is represented and maintained, enabling the system to track progress, reconcile concurrent updates, preserve provenance, isolate failures, and recover or adapt when execution deviates from plan. Together, these three dimensions transform graph structures from static representations into operational mechanisms for organizing and governing the execution of agent systems, thereby providing a structural foundation for `System Intelligence`.

Generally, we make the following contributions:

- Section 2 defines the core concepts of Individual Agents and Agent Systems. It characterizes an Individual Agent through its Foundation Model, Agent Harness, Agent Loop, and local runtime state, and an Agent System through its agent team, shared resources, environment, coordination mechanisms, and system-level state.
- Section 3 traces the evolution from Model Intelligence to Individual Intelligence and establishes the need for System Intelligence. It clarifies the roles of foundation-model development, Prompt and Context Engineering, and Harness and Loop Engineering, and identifies the structural limitations that motivate Graph Engineering.
- Section 4 formulates Graph Engineering as a system-level engineering paradigm and organizes the field into three interconnected views: `Task Organization` for structuring tasks, dependencies, and execution processes; `Agent Coordination` for organizing heterogeneous components and their collaboration; and `Runtime State Management` for tracking runtime states, supporting recovery, and enabling adaptation.

- Sections 5 and 6 outline open challenges and future research directions toward ontology engineering, dynamic and self-evolving graph systems, and graph-native agent operating systems.
- Section 7 synthesizes representative benchmarks, datasets, and executable environments across Model, Individual, and System Intelligence. It further identifies evaluation principles and open challenges concerning structural fidelity, operational correctness, system evolution, and governance.
- Section 8 surveys representative open-source libraries and engineering systems across the three intelligence levels. It examines how existing software stacks support model development, persistent agent runtimes, and multi-component orchestration, while highlighting gaps in interoperability, cross-run structural evolution, and state provenance.
- Section 9 reviews applications of Graph Engineering across software engineering, scientific discovery, healthcare, enterprise workflows, general-purpose digital agents, and social and economic simulation. It shows that Work Organization, Agent Coordination, and Runtime State Management are increasingly common in practice, whereas persistent System Evolution remains limited.

2 Preliminaries

We first clarify two fundamental concepts underlying Graph Engineering: the *Individual Agent* and the *Agent System*. The former defines the capabilities and operational mechanism of an autonomous entity, whereas the latter describes how multiple agents and supporting components are organized into a dynamic intelligent system.

2.1 Individual Agent

An Individual Agent is an autonomous computational entity that perceives its environment, makes decisions, executes actions, and adapts its behavior according to feedback. It consists primarily of a *Foundation Model* and an *Agent Harness*. The Foundation Model serves as the cognitive core, providing capabilities such as language understanding, reasoning, planning, and content generation. The Agent Harness extends these intrinsic capabilities through interfaces for perception and context construction, memory and knowledge access, tool invocation, reusable skills, and runtime governance.

These components are organized over time by an iterative Agent Loop, which repeatedly performs perception, reasoning, action, feedback processing, and state update. An Individual Agent can therefore be abstracted as

$$\mathcal{A}_i = \mathrm{Loop}(\mathcal{F}_i, \mathcal{H}_i; s_i^t), \tag{1}$$

where $\mathcal{F}_i$ denotes the Foundation Model, $\mathcal{H}_i$ denotes the Agent Harness, and s_i^t denotes the runtime state of agent i at time t. In this formulation, the Foundation Model determines the agent's intrinsic cognitive capabilities, the Harness determines the resources and action spaces it can access, and the Loop determines how these capabilities are continuously employed.

2.2 Agent System

An Agent System extends the Individual Agent abstraction to a collection of agents that operate through shared resources, external environments, and coordination mechanisms. At time t, an Agent System can be represented as

$$\mathcal{S}^t = \left(\mathbb{A}^t, \mathcal{R}^t, \mathcal{E}^t, \mathbf{\Pi}^t, \mathbf{x}^t\right), \tag{2}$$

where the system consists of the following elements:

- **Agent Team $\mathbb{A}^t$:** A collection of Individual Agents, each with its own Foundation Model, Harness, Agent Loop, and local runtime state. Agents may assume different roles, possess different capabilities, and undertake different tasks.
- **Shared Resources $\mathcal{R}^t$:** Resources and services accessible to multiple agents, including tools, model services, memory, knowledge bases, verifiers, and human support. Each agent accesses these resources through its Harness.

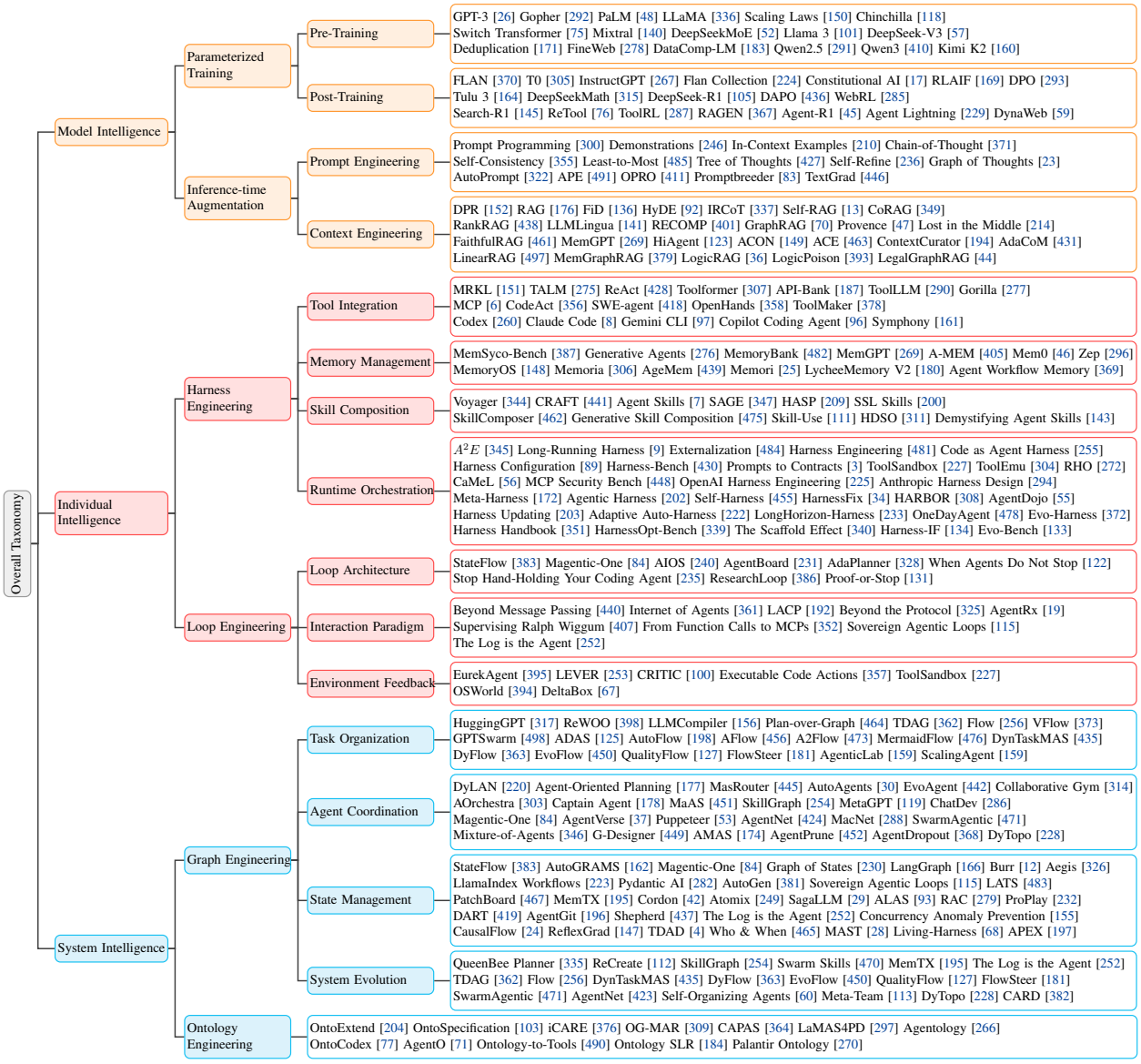

Figure 2 : A Taxonomy of Evolving Techniques in the Era of LLM Agents.

- **Environment $\mathcal{E}^t$:** The external environment that the Agent System perceives and acts upon. It provides observations and feedback and evolves in response to agent actions.
- **Coordination Mechanisms $\mathbf{\Pi}^t$:** Mechanisms that determine how agents assign tasks, exchange information, integrate results, resolve conflicts, and handle failures.
- **System State $\mathbf{x}^t$:** The system-level runtime information, including task progress, shared results, agent availability, resource status, environmental changes, and failure records. Unlike the local state s_i^t of an Individual Agent, $\mathbf{x}^t$ describes the operational condition of the entire system at time t.

The behavior of an Agent System therefore depends not only on the capabilities of its Individual Agents, but also on how shared resources are used, how agents coordinate with one another, and how the system state evolves throughout execution.

3 From Model Intelligence to Individual Intelligence

LLM-based intelligent systems have increasingly evolved from improving problem solving within individual inference processes toward constructing autonomous systems capable of sustained goal pursuit, external resource use, and

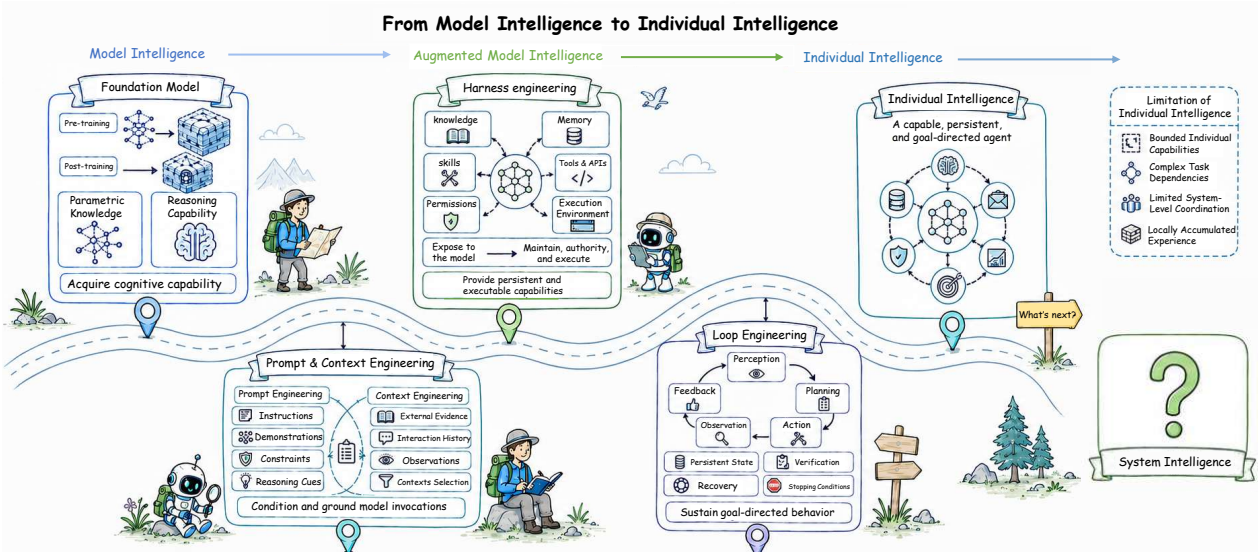

Figure 3 : From model intelligence to individual intelligence. Foundation-model capability is progressively transformed through task conditioning, persistent execution support, and feedback-controlled interaction into a capable, persistent, and goal-directed agent. The limitations of individual intelligence motivate the subsequent transition toward system intelligence.

environmental interaction. In the first stage, pre-training and post-training encode knowledge and general reasoning capabilities into model parameters, while Prompt Engineering and Context Engineering guide these capabilities toward effective use in specific tasks and contexts, giving rise to **Model Intelligence**. However, such intelligence remains bounded by relatively self-contained inference processes, limiting the model's ability to maintain persistent state, perform external actions, and continuously adapt to environmental feedback. To overcome these limitations, *Harness Engineering* connects the model to external knowledge, memory, tools, skills, and execution environments, thereby expanding its accessible and executable capabilities. *Loop Engineering* further organizes model reasoning and external capabilities into persistent cycles of planning, action, observation, verification, and adaptation. Through this transition, intelligence extends beyond problem solving within a given context toward **Individual Intelligence**, whereby an agent can use resources over time, adapt to its environment, and autonomously pursue goals. Fig. 3 summarizes this progression from the parametric capabilities of foundation models, through inference-time capability activation, external capability extension, and closed-loop execution, to individual intelligence.

3.1 Foundation Models: Establishing Model Intelligence

As LLMs have become increasingly capable [26, 48, 259], they have acquired general knowledge, reasoning, and problem-solving abilities that can be transferred across a wide range of tasks. These capabilities are largely encoded in model parameters through large-scale pre-training and subsequent Post-training [57, 101], forming the internal capability base of Model Intelligence. We refer to this process as *parameter-level capability development*, which typically involves two major stages: Pre-training and Post-training. Pre-training establishes a broad and reusable capability base, while Post-training further shapes how these capabilities are expressed and extends them toward desired behaviors and more complex task capabilities.

3.1.1 Pre-training

The development of Pre-training has been largely guided by scaling laws, which show that model performance improves as model size, training data, and computational resources are increased in a balanced manner rather than through parameter growth alone [118, 150]. Under this scaling paradigm, representative models such as GPT-3 [26], Gopher [292], LLaMA [336], Llama 3 [101], and DeepSeek-V3 [57] progressively strengthened general knowledge and problem-solving capabilities through larger-scale and more effective training. Its effectiveness depends on how scaling is realized through data, model architecture, and training strategy. High-quality and carefully curated data improve the quality and efficiency of knowledge acquisition [171, 183, 278]. Scalable architectures, including dense models and sparse

mixture-of-experts models such as Switch Transformer [75], Mixtral [140], and DeepSeekMoE [52], determine how model capacity can be expanded under practical computational constraints. Training strategies further determine how data scale, model capacity, and computational resources are allocated and coordinated during optimization. Advances in these areas have progressively strengthened the general capability base established during Pre-training.

3.1.2 Post-training

Post-training further updates model parameters so that the general capabilities acquired through Pre-training can be expressed as more controllable and reliable behaviors, while also developing capabilities for increasingly complex tasks [160, 291]. Based on their primary roles in modern training pipelines [164, 410], their development can be broadly discussed along three directions: Supervised Fine Tuning (SFT), Preference Alignment, and Reinforcement Learning (RL) for capability development. SFT evolved from early instruction tuning methods such as FLAN [370], T0 [305], and FLAN-PaLM [49] toward larger and more diverse instruction collections such as the Flan Collection [224], improving instruction following, response formatting, reasoning patterns, and task adaptation [61]. Preference Alignment introduced explicit human or model feedback: InstructGPT [267] established the influential RLHF pipeline, Constitutional AI [17] and RLAIF [169] extended alignment toward AI-generated principles and feedback, and DPO [293] simplified preference optimization by directly learning from preference pairs. More recently, RL has increasingly shifted from preference alignment toward direct capability development through verifiable rewards and environmental feedback. DeepSeekMath [315] introduced GRPO for mathematical reasoning, while DeepSeek-R1 [105] demonstrated that large-scale outcome-based RL can induce extended reasoning behaviors; subsequent methods such as DAPO [436], Dr. GRPO [221], and GSPO [477] further improve the stability and effectiveness of reasoning-oriented RL. This paradigm has also expanded toward *agentic RL*: Search-R1 [145] trains models to interleave reasoning with search, while ReTool [76], ToolRL [287], and ToRL [78] extend RL toward tool-integrated reasoning. WebRL [285], RAGEN [367], and WebAgent-R1 [374] further extend RL to interactive and multi-turn agent trajectories, while Agent Lightning [229] and DynaWeb [59] explore more general and scalable training frameworks for agents interacting with external environments. Modern Post-training pipelines therefore combine demonstrations, preference signals, verifiable rewards, and interaction feedback to jointly improve instruction following, behavioral alignment, reasoning, and agentic capabilities.

Together, Pre-training and Post-training determine the knowledge, reasoning abilities, and behavioral capabilities available to a model at the parameter level. However, a general capability base does not automatically translate into effective performance on a specific task. At inference time, the model still requires appropriate task descriptions, behavioral constraints, and task-relevant information to identify and apply the capabilities needed for the current problem.

3.2 Prompt and Context Engineering: Eliciting and Conditioning Model Intelligence

Pre-training and Post-training establish general capabilities at the parameter level, but these capabilities do not automatically translate into effective performance on specific tasks. Without modifying model parameters, Prompt Engineering and Context Engineering adapt these capabilities by shaping the control signals and information environment available at inference time. Prompt Engineering primarily concerns how tasks and expected behaviors are specified, whereas Context Engineering concerns what task-relevant information is provided and how it is organized and maintained. The former determines what the model should do and how it should approach the task, while the latter supplies the knowledge, evidence, and working state needed to complete it.

3.2.1 Prompt Engineering

The development of Prompt Engineering can be broadly characterized by three directions: task specification, reasoning organization, and automatic optimization. Task specification uses instructions, demonstrations, constraints, and output formats to support zero-shot and few-shot adaptation [26, 210, 246, 300]. Reasoning organization further structures how models solve complex problems: Chain-of-Thought [371] introduces intermediate reasoning, Self-Consistency [355] aggregates multiple reasoning paths, and Least-to-Most [485] decomposes difficult problems, while Tree of Thoughts [427], Graph of Thoughts [23], and Self-Refine [236] extend reasoning toward search and iterative refinement. Automatic prompt optimization moves prompt design from manual construction toward systematic search and improvement. Representative methods progress from AutoPrompt [322], APE [491], and OPRO [411] to planning-, evolutionary-, and program-level optimization [83, 108, 264, 354, 446]. More recently, GEPA [2] uses execution

trajectories and natural-language reflection to evolve prompts from task feedback. Overall, Prompt Engineering has expanded from specifying tasks, to organizing reasoning, and ultimately to optimizing the control interface itself.

3.2.2 Context Engineering

Context Engineering extends this focus to the broader information environment used during task execution, including context acquisition, processing, and management. Context acquisition has progressed from dense retrieval [152] and retrieval-augmented generation [136, 176] toward retrieval coupled with reasoning. HyDE [92], IRCoT [337], Self-RAG [13], and CoRAG [349] progressively integrate query transformation, iterative retrieval, and reflection, while recent work further studies when retrieval should occur during reasoning and how retrieval itself can become an agentic process [44, 106, 247, 379, 392, 461]. Context processing improves the relevance, compactness, and structure of acquired information through ranking [438], compression [141, 401], pruning [47], and restructuring [70]. Recent methods such as SARA [146] and BRIEF-Pro [104] further improve information-preserving compression under constrained context budgets, while Lost in the Middle [214] shows that longer context alone does not guarantee effective information use. Context management maintains useful working information as execution progresses. MemGPT [269] introduced explicit hierarchical context management, followed by hierarchical and adaptive approaches such as HiAgent [123], ACON [149], ACE [463], ContextCurator [194], and AdaCoM [431]. Context as a Tool [215] further treats context maintenance as an explicit agent action, enabling proactive compression during long-horizon execution.

Prompt Engineering and Context Engineering together constitute an inference-time mechanism for adapting model capabilities. Prompt Engineering establishes the control structure for task execution, while Context Engineering provides and maintains the task-relevant information base. Rather than altering the general capabilities encoded in model parameters, they determine which capabilities are invoked, how reasoning unfolds, and what information conditions model outputs, thereby translating general capabilities into task-specific behavior.

3.3 Harness Engineering: Orchestrating Agent Capabilities

Model Intelligence takes the model call as its basic unit of operation. Foundation model training establishes general capabilities at the parameter level, while Prompt Engineering and Context Engineering adapt them to specific tasks. However, a model call alone cannot maintain persistent resources, execute external operations, or sustain interaction with an environment over time. Extending Model Intelligence toward *Individual Intelligence* therefore requires persistent and executable capabilities that remain available across calls. Harness Engineering provides and manages these capabilities, while Loop Engineering organizes how they are repeatedly invoked and adapted during task execution. Recent work increasingly treats the harness as the runtime layer surrounding the model, connecting memory, tools, skills, execution environments, state, verification, and other supporting mechanisms into an operational agent system [9, 54, 255, 481, 484].

3.3.1 Tool Integration

Many mechanisms now associated with Harness Engineering appeared before the term itself became widely established. Early systems such as MRKL [151], TALM [275], and Toolformer [307] connected models to external tools. Tool use subsequently expanded toward large API ecosystems through API-Bank [187], ToolLLM [290], and Gorilla [277], while MCP [6] provides a standardized interface to external tools and data sources. CodeAct [356], SWE-agent [418], and OpenHands [358] further extend execution to code, files, shells, browsers, and computing environments, moving external capability access from function invocation toward richer agent-computer interaction [62, 193].

3.3.2 Memory Management

Persistent memory allows agents to retain information and experience beyond individual model calls. Generative Agents [276], MemoryBank [482], and MemGPT [269] established early mechanisms for persistent and long-term memory, while later systems such as A-MEM [405], Mem0 [46], Zep [296], and MemoryOS [148] improve memory organization, consolidation, and reuse. Recent work increasingly moves memory from a passive storage component toward an actively managed agent capability. AgeMem [439] integrates short- and long-term memory operations into the agent policy, allowing the model to decide when to store, retrieve, update, summarize, or discard information, while graph-based systems such as GAM [384] and HeLa-Mem [493] organize evolving experiences through explicit

relational structures. MAGE [41] further treats memory as execution-state management for long-horizon tasks, supporting state reconstruction and recovery, while Text2Mem [350] introduces typed and executable memory operations for more controllable memory management. Other recent work explores efficient consolidation, filesystem-based persistent memory, and the reliability of memory addition and deletion [397, 458, 489]. Together, these developments shift memory engineering from storing past information toward actively organizing, governing, and maintaining reusable experience across extended execution.

3.3.3 Skill Composition

Beyond individual tools and memories, skill-based methods externalize successful procedures as reusable capabilities. Voyager [344] introduced an executable skill library accumulated from experience, while Agent Workflow Memory [369] reuses recurring action workflows and Agent Skills [7] packages instructions, scripts, and supporting resources into reusable procedural capabilities. Subsequent methods such as SAGE [347], HASP [209], SkillComposer [462], and Skill-Use [111] improve skill construction, composition, evolution, and invocation. More recent work increasingly treats the skill library itself as an adaptive engineering object. SkillX [343] automatically constructs hierarchical skill knowledge bases from trajectories, while SkillOpt [425] uses execution feedback to systematically optimize reusable skill artifacts. Anything2Skill [271] compiles heterogeneous external knowledge into reusable procedural skills, extending capability acquisition beyond direct trajectory reuse. As skill libraries grow, SkillOps [281] and SkillWiki [129] address library-level maintenance, provenance, governance, and lifecycle evolution, while SkillZip [16] and related approaches [468] reduce the runtime and maintenance cost of repeatedly using large skill artifacts. These developments expand skill engineering from acquiring individual reusable procedures toward constructing, optimizing, maintaining, and evolving persistent skill ecosystems.

3.3.4 Runtime Orchestration

As external capabilities become richer, Harness Engineering increasingly concerns how these resources are organized, governed, verified, and improved as a runtime system. Anthropic's long-running agent harness [9, 294], AI Harness Engineering [481], What Makes a Harness a Harness [54], Code as Agent Harness [255], and Harness-Bench [430] make this surrounding runtime an explicit research object and clarify its responsibilities and boundaries. In this view, Context Engineering determines what information is presented to a model call, whereas Harness Engineering maintains the persistent resources, interfaces, and execution environments through which information and external capabilities remain available across calls.

Runtime orchestration also introduces configuration, governance, verification, and optimization concerns. ToolEmu [304], ToolSandbox [227], AgentDojo [55], CaMeL [56], and MCP Security Bench [448] study failures, security risks, and control mechanisms, while harness configuration [89] and contract-based validation [3] address configuration and runtime guarantees. These engineering principles are increasingly reflected in widely used coding agents such as Codex [225, 260], Claude Code [8, 294], Gemini CLI [97], and GitHub Copilot coding agent [96]. More recent work treats the harness itself as an optimization target: Meta-Harness [172], Agentic Harness Engineering [202], Self-Harness [455], HarnessFix [34], HARBOR [308], and Retrospective Harness Optimization [272] explore search, adaptation, diagnosis, repair, and feedback-driven improvement. Related work further evaluates harness effects and optimization [134, 203, 339, 340, 353], while Adaptive Auto-Harness [222], LongHorizon-Harness [233], OneDayAgent [478], and Evo-Harness [372] extend adaptation toward open-ended and long-horizon execution. Harness Handbook [351] further addresses the understandability and maintainability of increasingly complex harnesses. Harness Engineering therefore concerns not only what external capabilities an agent can access, but also how the runtime surrounding the model is structured, governed, maintained, and improved over time.

3.4 Loop Engineering: Enabling Iterative Agent Execution

Harness Engineering establishes the capability space available to an agent, whereas Loop Engineering organizes how the agent moves through that space over time. A harness provides persistent resources, executable tools, validation mechanisms, and controlled environments, together with the interfaces and permissions governing their use. However, it does not by itself determine how an active task should proceed after each execution. For example, when a test fails, the harness can return the failure log, but the loop can decide whether to revise the implementation, inspect a dependency, invoke another capability, recover an earlier state, request assistance, or terminate the task. We therefore

define *Loop Engineering* as the engineering of a bounded, stateful, and feedback-driven process that coordinates agent operation until the goal is supported by sufficient evidence or continued execution is no longer justified [131, 234, 386]. Its defining property is not the repetition of model calls, but the continuous use of execution outcomes to control the subsequent trajectory of the task.

We examine Loop Engineering through three coupled aspects: *Loop Architecture*, *Interaction Paradigm*, and *Environment Feedback*. Loop Architecture defines the control structure through which goals, task states, operations, verification, and termination are organized. Interaction Paradigm determines how relevant information is exchanged among the loop controller, model, harness, and supervisory actors. Environment Feedback grounds this process in external consequences by transforming observed state changes into evidence for subsequent decisions. Together, these aspects form a closed process in which control decisions generate operations, operations produce environmental consequences, and the resulting observations return to update loop control.

3.4.1 Loop Architecture

Loop Architecture describes the main components of a loop architecture and how they jointly keep goal-directed execution coherent and bounded. A loop is initialized with a goal, acceptance criteria, and terminal conditions, after which a controller maintains the operational state of the task and tracks which requirements remain unresolved. Planning and decomposition mechanisms organize these requirements into executable operations, while progress assessment and verification determine whether an operation has produced a meaningful state change. The architecture must also specify how the loop responds to failure, including revising the plan, selecting another capability, recovering an earlier state, escalating the task, or terminating execution. ResearchLoop [386] represents task contracts, evidence objects, claim ledgers, and closeout conditions as durable control state, allowing research activities to advance only when their evidence requirements are satisfied. Proof-or-Stop [131] similarly permits lifecycle transitions only when fresh and mechanically verifiable evidence satisfies the relevant gate. Complementary work examines when additional search has become redundant [333], while analyses of infinite agentic loops [121] show that progress checks, resource limits, and explicit stopping conditions are necessary to prevent unbounded feedback paths. Therefore, Loop Architecture determines not only how execution continues, but also when continuation remains justified.

3.4.2 Interaction Paradigm

Building on this control structure, *Interaction Paradigm* explains how task state, action requests, observations, and supervisory signals are exchanged across iterations. At each step, the loop communicates the current goal, task state, unresolved requirements, and available operations to the model; the model returns proposed decisions or action intents; and the harness returns execution observations, validation results, and error conditions. These exchanges must preserve sufficient continuity for later decisions to be interpreted relative to earlier actions and outcomes. ResearchLoop [386] maintains this continuity through persistent task contracts, claim ledgers, and evidence records. Sovereign Agentic Loops [115] formalizes the action boundary by representing model outputs as structured intents that can be checked against system state and policy before execution. Interaction may also introduce diagnostic and supervisory feedback. AgentRx [19] transforms execution trajectories into validation records that localize critical failure steps, whereas Supervising Ralph Wiggum [407] introduces metacognitive supervision when repeated refinement becomes stagnant. The harness provides communication interfaces, memory mechanisms, and execution records; the loop determines which information is relevant to the current task and how that information changes subsequent control decisions.

3.4.3 Environment Feedback

After these interaction pathways have been established, *Environment Feedback* examines how external execution outcomes return to the loop as decision-relevant feedback. An authorized operation may invoke a tool, execute a program, modify an artifact, or change an external system, but the intended effect, actual state transition, and observed outcome may differ. A reliable loop must therefore determine which observations are required, whether they reflect the current environment state, how they relate to the task goal, and whether they justify accepting progress or initiating another operation. Proof-or-Stop [131] binds lifecycle transitions to evidence associated with the current source state, preventing stale or unsupported results from being treated as task completion. Sovereign Agentic Loops [115] likewise checks proposed actions against true system state and policy before allowing real-world execution. Failed verification may

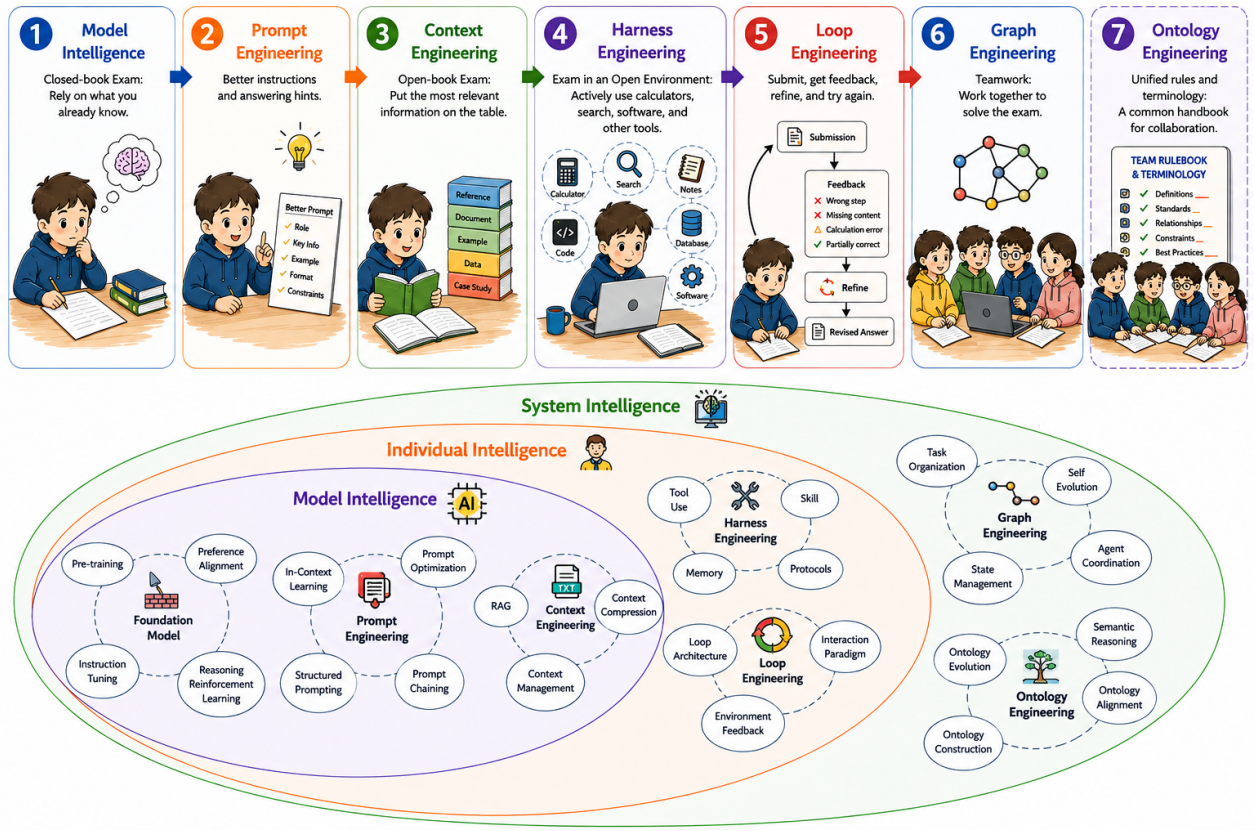

Figure 4 : An Illustrative Conceptualization of System Intelligence and Its Related Technologies. The exam analogy depicts the progression from Model, Prompt, and Context Engineering to tool-enabled Harness Engineering, feedback-driven Loop Engineering, team-oriented Graph Engineering, and ontology-based collaboration. The lower panel summarizes representative technologies associated with these layers.

cause the loop to retry, revise its plan, invoke another capability, restore a previous state, or escalate the task. Environment Feedback therefore refers not to the execution environment itself, which is supplied by the harness, but to the use of environmental state changes and observations as evidence governing the continuation of the loop.

Harness Engineering and Loop Engineering consequently address the complementary requirements of *Individual Intelligence*. Harness Engineering determines what persistent and executable capabilities an individual agent can access and under what conditions they can be used. Loop Engineering organizes those capabilities into a bounded, goal-directed process in which actions, observations, feedback, and termination decisions remain connected across time. Together, they extend call-level Model Intelligence into the sustained behavior of an individual agent and thereby establish *Individual Intelligence*.

3.5 Limitations of Individual Intelligence

Despite the advances in Harness and Advances in Harness and Loop Engineering have enabled agents to exhibit individual intelligence, allowing them to pursue goals autonomously through sustained reasoning and interaction with their environment. However, since individual intelligence is typically organized around *a single agent* and its execution loop, it still faces several fundamental limitations when applied to complex real-world tasks:

❶ **Scheduling parallel and interdependent tasks:** Real-world tasks often contain subtasks that depend on one another or can be carried out in parallel. A single-agent loop, however, tends to compress them into a serial execution trace [93, 126]. This makes scheduling implicit, wastes the efficiency of parallelism, and makes failure location difficult. For example, in a software fault diagnosis task, log analysis, failure reproduction, and code inspection can often proceed in parallel as relatively independent branches, whereas repair and testing depend on their results. A single agent, however, tends to serialize these branches within a single execution loop, losing the efficiency of parallelism. Moreover, wrong

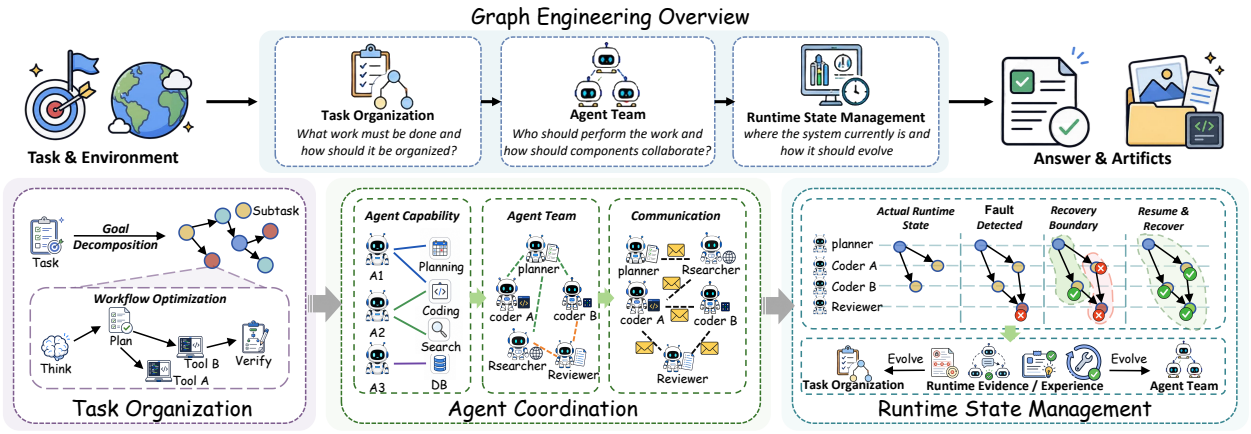

Figure 5 : Overview of Graph Engineering. Task Organization structures the objective into explicit subtasks and executable workflows; Agent Coordination matches capabilities to work, defines team topology, and routes communication among agents; Runtime State Management records execution states, detects and localizes anomalies, and supports recovery and structural updates. Together, these coupled graph views organize work, coordinate agents, and use runtime evidence to evolve the system toward reliable task completion.

intermediate results may be propagated to subsequent steps, making the faulty stage difficult to localize.

❷ **Integrating specialized expertise and verification:** Many complex tasks require specialized expertise or independent verifiers. Although a single agent can use tools or call specialist models, these capabilities remain coordinated within the same control loop rather than organized into stable and independent roles [28, 88, 288]. This can lead to role confusion and confirmation bias. For example, when the same agent writes and evaluates code, it may mistake its own judgment that the code is correct for evidence that it is actually correct, even when prompts assign it different roles.

❸ **Maintaining persistent state and handling failure recovery:** An individual's context is not an organized or persistent state. Once an error enters the execution loop, it can be carried through later steps, making it difficult to repair only the affected parts or to recover in a way that can be traced and checked [28, 110, 155]. For example, in long-running web or coding tasks, a small mistake made early may remain hidden until the task fails near the end. By then, it is often difficult to determine and localize where the error first appeared.

4 Graph Engineering: From Individual Intelligence to System Intelligence

4.1 Overview of Graph Engineering

Despite advances in individual intelligence, the unit of **Individual Intelligence** still faces inherent limitations in scheduling parallel and interdependent tasks, integrating heterogeneous capabilities, and maintaining runtime state. This motivates the next stage of intelligence toward **System Intelligence**, in which multiple components with complementary capabilities form an adaptive whole in pursuit of a shared goal [87, 154, 159].

However, system intelligence does not arise from the mere aggregation of agents and other intelligent components; rather, it depends on how the relationships among tasks, components, and runtime states are explicitly represented, constrained, and optimized. Specifically, a system must organize objectives into decomposable and schedulable task structures, coordinate heterogeneous components according to their capabilities and roles, and maintain persistent and recoverable runtime state throughout execution. At its core, *system intelligence requires the systematic governance of relationships among tasks, components, and runtime states*.

To address this, graphs provide a natural structure for modeling the system-level relationships, as shown in Fig. 5. *First*, graphs organize tasks through objective decomposition, dependency modeling, and workflow refinement, transforming complex objectives into schedulable and executable operations [91, 158, 217, 280, 416, 466]. *Second*, graphs can coordinate intelligent components by representing operational topologies and communication patterns, enabling heterogeneous components to collaborate effectively [69, 79, 205, 241, 310, 331, 390, 426, 433]. *Third*, graphs can support runtime state management by recording events, dependencies, and state transitions, converting operational information scattered across contexts and logs into auditable and recoverable system states [35, 41, 467]. To this end,

we introduce **Graph Engineering** as a structure-centered engineering foundation for system intelligence: it uses graph structures as the core substrate for externalizing relationships among tasks, components, and runtime states, thereby supporting system-level organization, coordination, monitoring, recovery, and optimization.

In the following section, we review existing approaches that leverage graphs to organize tasks, coordinate intelligent components, and manage execution states. We further discuss how system evolution leverages execution feedback and state evidence to iteratively improve the structure of Graph Engineering and enable the continual evolution of system intelligence.

4.2 Task Organization: Structuring What to Do

The first challenge in building system intelligence is to transform a high-level objective or task stream into an organized set of subtasks and operations, enabling intelligent components to perform interdependent actions rather than isolated local tasks [39, 80]. However, subgoals may depend on one another, and their execution may involve parallel branches, verification steps, and dynamic replanning. Relying solely on context makes it difficult to maintain a clear global task structure and determine which operations should be performed. To address this challenge, existing work externalizes task decomposition and executable operations as graph structures, transforming task organization from implicit reasoning into a schedulable, optimizable, and revisable system structure, as shown in Fig. 6. We next discuss how graph structures provide the foundation for goal decomposition and workflow optimization.

4.2.1 Goal Decomposition

System intelligence requires explicit goal decomposition so that intelligent components can perform coordinated, interdependent actions. However, complex user objectives or task streams [85, 409] are difficult to organize within context alone: their subtasks may depend on one another, some steps may run in parallel, and the execution plan may need revision as intermediate results arrive. Graph-based task decomposition addresses this challenge by representing an objective as a graph of subgoals and dependencies, where nodes denote subtasks or intermediate goals and edges encode precedence, data, or logical relations. This explicit structure supports the scheduling of parallel and dependent branches and provides a basis for workflow refinement and component coordination.

Early work began by making task decomposition and subtask dependencies explicit, rather than leaving them implicit within an execution loop. HuggingGPT [317] decomposes multimodal user requests into subtasks and routes them to specialized models, using dependency relations to determine their execution order. ReWOO [398] decouples reasoning from tool execution and observations through variable references, making dependencies among planned tool calls explicit.

To enable better task scheduling, subsequent studies represent these task dependencies as explicit and schedulable graphs. LLMCompiler [156] compiles function-calling plans into a dataflow DAG, so that ready nodes can be dispatched in parallel once their upstream dependencies are satisfied. Plan-over-Graph [464] directly studies planning over task graphs and focuses on generating parallelizable agent schedules under dependency constraints. These works not only make the task dependencies explicitly interpretable but also operational for scheduling and coordination. TDAG [362] and Flow [256] further relax the assumption that the task graph is fixed before execution. They show that task decomposition can be dynamically refined according to intermediate results, and in multi-agent settings, such evolving task graphs can also drive agent generation, task assignment, and parallel collaboration [435].

In short, Goal Decomposition Graph divides the objective goal into explicit, schedulable sub-goal graphs. It defines the structured objective space over which later workflow construction and execution adaptation operate.

4.2.2 Workflow Optimization

After the subgoals and their dependencies are known, system intelligence still needs to transform them into concrete computational operations, such as LLM calls, specialized agents, retrieval modules, tools, memory operations, aggregators, and verifiers. However, the operations space is large and must be explicitly structured to construct an effective workflow. To address this challenge, existing methods use graph structures to compile decomposed tasks into executable workflows, where nodes represent concrete operators and edges encode the dependencies needed for scheduling, coordination, and verification. This process transforms task organization from a descriptive decomposition into an executable structure that can be optimized.

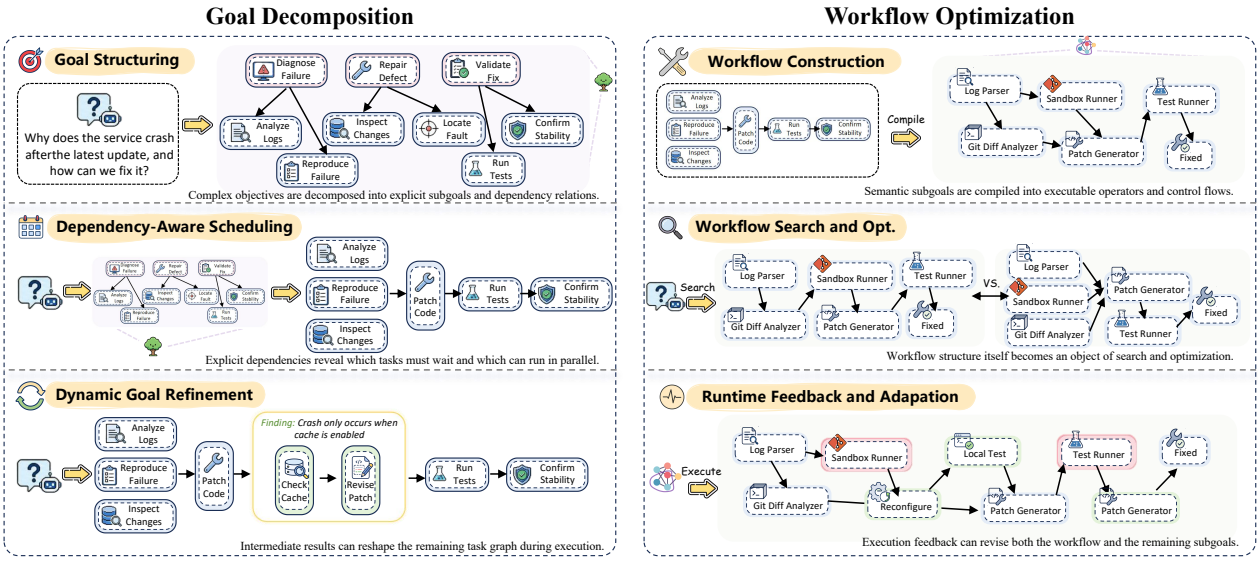

Figure 6 : Overview of Task Organization. Goal Decomposition translates a high-level objective into explicit subtasks, exposes their dependencies for scheduling, and refines the remaining task graph using intermediate execution results. Workflow Optimization compiles semantic subgoals into executable workflows, searches and optimizes alternative control flows, and adapts execution in response to runtime feedback. These mechanisms specify what work must be accomplished, how it should be operationalized, and how the work structure evolves during execution.

A group of studies treats agentic workflows as optimizable graph structures. GPTSwarm [498] represents language-agent systems as computational graphs and optimizes both node behavior and edge connections. ADAS [125] automates the design of agentic systems by searching over code-defined workflows, where graph semantics are expressed through executable program structures. AutoFlow [198] and AFlow [456] formulate workflow generation as an automatic search problem, reducing reliance on manually designed agent pipelines. In particular, AFlow uses LLM-guided search over executable workflow code, making the workflow structure itself the object of optimization. Later works refine different parts of this workflow search space. A2Flow [473] learns abstraction operators from demonstrations instead of assuming a fixed operator library, allowing both node semantics and graph topology to evolve. Mermaid-Flow [476] introduces a structured Mermaid-based intermediate representation and safety-constrained evolutionary programming, improving the readability, validity, and controllability of generated workflows. VFlow [373] incorporates domain-specific verifiers into the workflow search loop, showing how external feedback such as syntax checks, functional correctness, synthesizability, and hardware constraints can guide workflow discovery.

Despite the advances in static workflow optimization, these approaches remain insufficient for open-ended environments. Even well-designed task and workflow graphs may fail or propagate errors during execution due to incorrect intermediate results, tool failures, or ambiguous feedback. To address this limitation, recent approaches have developed dynamic mechanisms that adapt workflow graphs in response to real-time execution feedback. DyFlow [363] exemplifies this execution-adaptive paradigm. Instead of committing to a fixed workflow before execution, it uses intermediate feedback to dynamically generate and adjust subsequent operator subgraphs. In this sense, runtime adaptation can revise both the local workflow and the remaining subgoal structure. EvoFlow [450] maintains diverse workflow candidates during inference and evolves them on the fly, treating different workflow graphs as competing executable hypotheses. QualityFlow [127] introduces quality checking as a control mechanism for program synthesis, where the system dynamically selects whether to accept, debug, clarify, roll back, or continue based on intermediate quality signals. FlowSteer [181] further highlights that workflow structure can be modified inside the execution loop, rather than only optimized before deployment.

In summary, Task Organization provides a unified view of graph-based task and execution management in agentic systems. It shifts the design of LLM agents from implicit reasoning and acting to explicit work structures.

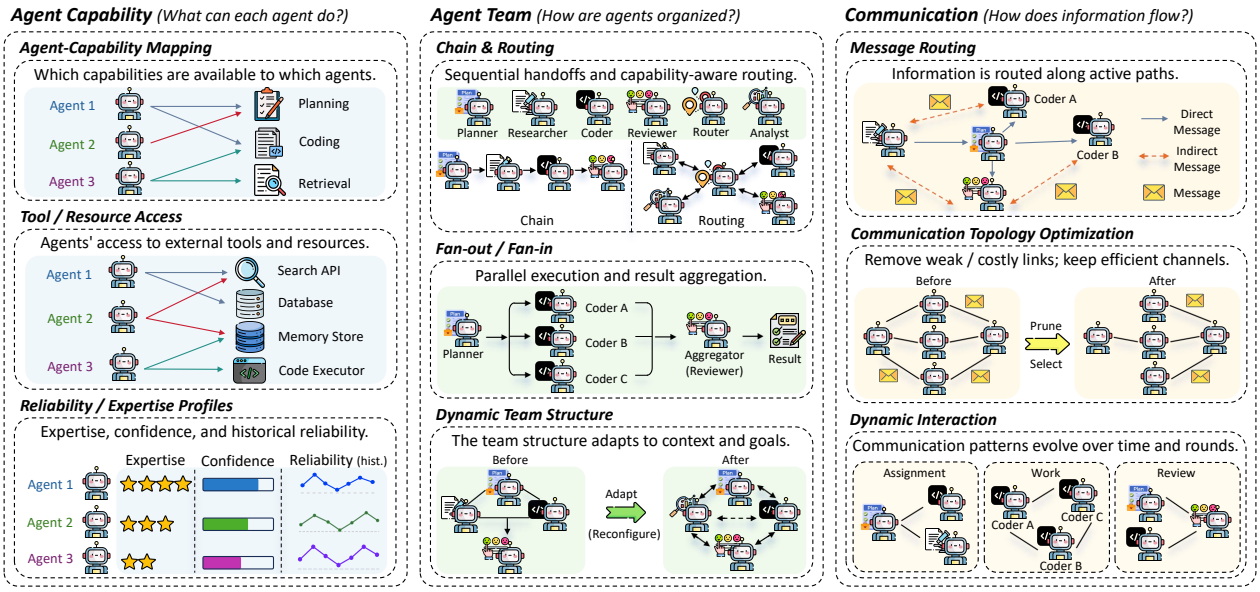

Figure 7 : Overview of Agent Coordination. The Agent Capability Graph maps agents to their capabilities and accessible resources; the Agent Team Graph organizes agents into task-dependent collaboration structures; and the Communication Graph specifies and adapts information flow among agents. These graphs determine who should perform the work and how agents collaborate during execution.

4.3 Agent Coordination: Structuring Who Works

The second challenge in building system intelligence is coordinating heterogeneous agents as a coherent system rather than invoking them within a single control loop. This requires identifying what different agents can do, assigning them appropriate roles and responsibilities, and adapting their interactions as execution unfolds [63, 191, 250, 454]. As shown in Fig. 7, *Agent Coordination* addresses these requirements through three connected functions that can be represented using graph structures [206, 302, 447, 486]. *Agent Capability Modeling* represents agents' skills, resources, permissions, and suitability for different tasks. *Agent Team Organization* arranges selected agents into task-dependent collaboration structures, specifying role assignments, delegation paths, and review responsibilities. *Multi-agent Communication* captures the runtime information exchange and feedback through which agents coordinate actions, evaluate intermediate results, and adapt subsequent execution. Together, these functions determine who should perform the work, how responsibilities should be organized, and how agents should interact as task conditions change.

4.3.1 Agent Capability Modeling

System intelligence requires heterogeneous work to be assigned to agents with suitable expertise and resources as the demands of complex tasks evolve. Because task stages are interdependent, a capability mismatch at one stage may delay parallel execution and compromise downstream results. The system must therefore maintain up-to-date information about each agent's skills, available resources, access permissions, and reliability. To address this challenge, graph structures make this information explicit: nodes represent agents, skills, tools, models, and other resources, while typed edges encode capability ownership, resource access, permissions, and reliability [22, 109]. This representation enables capability-aware task assignment and agent reconfiguration as execution conditions change [132, 400]. For example, in scientific discovery, literature analysis, experiment design, implementation, and independent verification can be assigned to suitable agents. If an agent loses access to a computing resource, the system can query the graph to identify a compatible replacement and reassign the affected task.

Existing methods often infer capability from task-specific behavior. DyLAN [220] estimates the contribution of candidate agents and retains those that are more useful for the current task, while Agent-Oriented Planning [177] assigns solvable and non-redundant subtasks to suitable agents. MasRouter [445] further learns to select collaboration modes, roles, and underlying models according to task difficulty and cost. These methods capture capability differences effec-

tively, but capability is mainly encoded in scores or routing policies rather than explicit and reusable relations [189].

Other methods represent capability through agent configuration. AutoAgents [30] creates specialized roles and collaboration plans for a given task, EvoAgent [442] generates diverse specialists through evolutionary operations, and AOrchestra [303] composes instructions, context, tools, and models to instantiate task-specific agents. Captain Agent [178] similarly recruits and reorganizes experts as new requirements emerge during interaction.

More recent work connects capability modeling with graph-based organization. SkillGraph [254] explicitly represents agent skills and uses them to guide the construction of communication topologies. MaAS [451] takes a broader approach by representing agents and operators within an agentic supernet and searching this space for suitable multi-agent structures. However, these representations are typically constructed for a particular task or orchestration process. A persistent and updateable graph representation would instead allow knowledge about agents' expertise, reliability, available resources, and access permissions to be queried, revised, and reused across tasks.

4.3.2 Agent Team Organization

System intelligence requires heterogeneous agents to be organized into a team that can execute interdependent work coherently. Capability modeling identifies which agents are suitable for particular tasks, but it does not determine task ownership, output handoffs, delegation paths, or review responsibilities. To address this challenge, these organizational relations can be represented as a graph, in which nodes denote agents, roles, or tasks, and typed edges encode assignment, delegation, supervision, verification, and reporting relations [31, 38, 173, 486]. By specifying each agent's position and responsibilities, this representation connects individual capabilities to an executable division of labor [113, 208, 274, 327, 459].

For tasks with clear stage dependencies, agents can be organized into a chain in which the output of one role becomes the input to the next. MetaGPT [119] structures software-development agents as an assembly line governed by standard operating procedures, while ChatDev [286] connects design, coding, and testing roles through a sequential chat chain. Such structures make execution order, role transitions, and responsibility boundaries explicit, although their paths are largely fixed before execution [124, 312].

When subtasks require different expertise, routing structures direct each unit of work to an appropriate agent. Magentic-One [84] uses an orchestrator to plan and delegate tasks, monitor progress, and replan after failures. WorkTeam [208] employs a supervisor that invokes specialized orchestrator and filler agents according to user intent, while AgentVerse [37] composes teams of experts according to task requirements. Routing supports specialized division of labor, but centralized designs may impose substantial planning and coordination burdens on the routing agent.

Tasks that benefit from parallel execution or diverse candidate solutions can instead adopt fan-out/fan-in structures. Work is distributed to multiple agents and their outputs are subsequently compared, aggregated, or synthesized. Mixture-of-Agents [346] uses a layered structure in which several agents generate candidate responses in parallel and agents in the next layer integrate them. MacNet [288] generalizes this branching and aggregation process through a directed acyclic graph, allowing multiple execution paths to converge at downstream nodes. These structures increase parallelism and reasoning diversity, but also incur additional communication, computation, and aggregation costs.

Static team structures become less effective when task requirements or agent performance change during execution [207]. Puppeteer [53] dynamically selects and sequences agents according to the current task state. AgentNet [424] removes the central controller and allows agents to adjust their connections and route tasks based on local expertise and context. Team organization can also be optimized during system construction. SwarmAgentic [471] jointly optimizes agent functions and collaboration patterns while generating candidate systems. Studies of self-organizing agents [60] further suggest that role specialization and shallow hierarchies can emerge without fully predefined assignments. These methods adapt team organization at different timescales, from design-time optimization to runtime reconfiguration.

Graph-based team organization can combine these structures within a single system. A coordinator may route subtasks to specialists, distribute selected tasks for parallel execution, aggregate their outputs, and pass the combined result through a chain of reviewers. The graph must therefore represent both stable responsibility relations and task-dependent structural changes, specifying who participates, what each participant is responsible for, and how work moves among them.

4.3.3 Multi-agent Communication

As task execution unfolds, system intelligence must coordinate information exchange among agents and prevent unreliable intermediate results from propagating downstream. Errors, conflicts, and missing information may require clarification, review, feedback, or human intervention [73, 273, 470]. To address this challenge, these runtime interactions can be modeled as a dynamic graph, where nodes represent agents or human participants and activated edges specify who communicates, what information is exchanged, and how it affects subsequent actions. Whereas team organization defines relatively stable roles and responsibilities, communication modeling captures the information flows and feedback relations that emerge during execution.

Communication serves not only to transfer results but also to detect and correct errors. Different agents can generate, evaluate, and revise an output, returning identified problems to the relevant execution stage. MAgICoRe [32] combines model-generated feedback with external stepwise reward signals to locate reasoning errors and iteratively refine candidate solutions through multi-agent interaction. This process forms a feedback loop among generation, evaluation, and revision rather than a one-way flow of information. Communication structure also determines how correct and incorrect information propagates, so adding more connections does not necessarily improve collaboration [316].

Communication structures can be constructed according to task requirements and optimized under multiple objectives. G-Designer [449] generates task-dependent communication graphs by considering candidate agents, performance, communication cost, and structural robustness. AMAS [174] selects interaction structures according to the current input, allowing different tasks to employ different communication patterns. Other methods reduce collaboration overhead by removing low-value relations. AgentPrune [452] eliminates redundant connections from a spatio-temporal message graph, while AgentDropout dynamically removes low-contribution agents and their communication edges across interaction rounds [368]. These methods indicate that communication modeling should determine not only whether information can be transmitted, but also which information paths are worth maintaining.

Runtime feedback can further be used to adapt subsequent communication. DyTopo [228] reconstructs sparse communication edges in each round by matching the information required by one agent with that available from others. CARD [382] incorporates environmental signals, including changes in model capabilities, tool availability, and computational resources, enabling communication structures to adapt during both training and execution. QueenBee Planner [335] extracts communication design knowledge from execution traces and evaluation results, converting it into structural rules that can be reused and revised in later tasks. These approaches extend communication optimization from one-time topology selection to a feedback-driven process informed by current conditions and previous outcomes.

Not all feedback can be generated reliably by agents. Tasks involving implicit preferences, specialized expertise, or high-risk actions may require humans to clarify requirements, correct errors, review outputs, approve actions, or assume control. Collaborative Gym [314] supports asynchronous and bidirectional interaction among humans, agents, and task environments, allowing human participation throughout execution. Graph-based communication modeling can represent humans as explicit participants, with edges denoting assistance requests, feedback, approval, and escalation. Humans are thus incorporated as active collaborators in the feedback loop rather than being limited to evaluating the final result.

Graph-based communication modeling should therefore be distinguished from the relatively stable representation of team organization. Team organization determines who participates and what responsibilities they assume. Communication modeling captures who needs to exchange information at a particular point in execution, how feedback is transmitted, and how that feedback changes subsequent actions.

4.4 Runtime State Management: Structuring How the System Operates

In an individual agent, runtime state can often remain local to its context, memory, and action history, as observation, decision, and execution are largely unified within a single locus of control. However, when intelligence is distributed across a system, execution is split across interdependent tasks and specialized agents, together with partial observations and external effects. Although *Task Organization* and *Agent Coordination* specify what should be executed and by whom, they do not by themselves maintain a reliable account of what has happened, which commitments remain valid, or how one state change affects later decisions. Without such an account, agents may act on inconsistent views, failures are difficult to localize, and valid progress is difficult to recover. To address this gap, we introduce *Runtime State Management*, which provides three complementary capabilities, as illustrated in Fig. 8: *State Recording* maintains

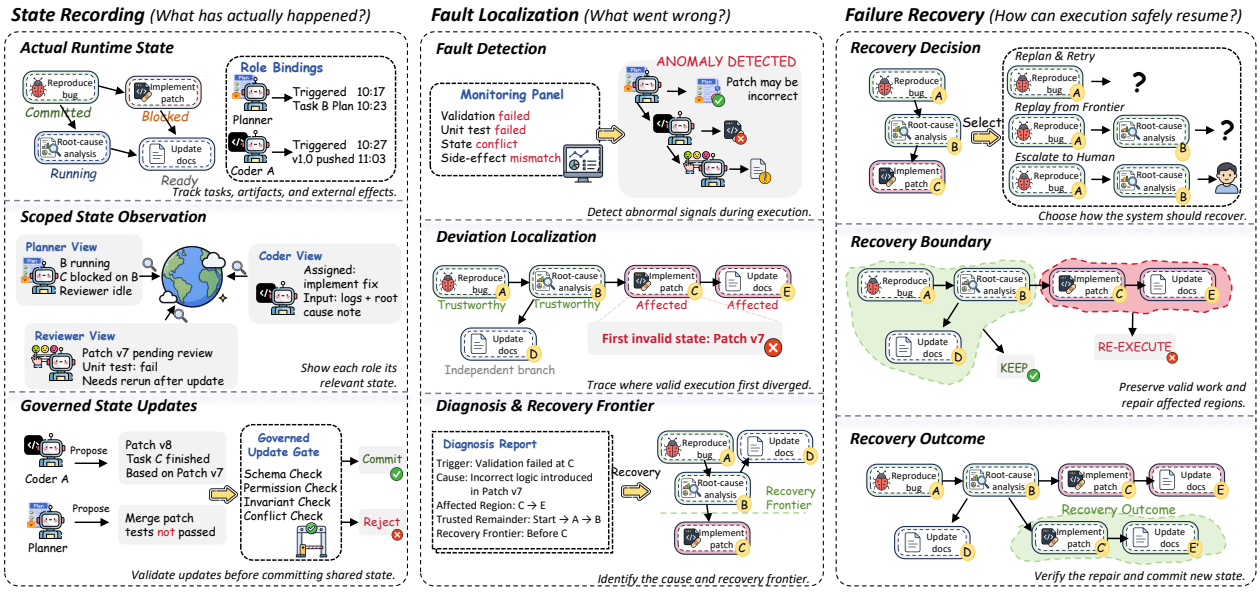

Figure 8 : Overview of Runtime State Management. Runtime State Management structures how to manage state by recording consistent and traceable runtime views, localizing failures from execution evidence, and recovering from validated states. These capabilities turn distributed execution histories into a reliable substrate for monitoring, diagnosis, recovery, and continual system evolution.

consistent and traceable runtime views, *Fault Localization* identifies deviations from intended execution, and *Failure Recovery* restores or redirects execution from validated states.

4.4.1 State Recording

Task Organization and Agent Coordination describe what should happen and who should act, but they do not record what has actually happened. During distributed execution, different agents and tools produce partial updates about progress, role bindings, commitments, shared facts, resources, and external effects. If these updates are not kept in a consistent and traceable form, system intelligence cannot maintain a reliable view of the current run or provide evidence for later diagnosis and recovery. To address these challenges, *State Recording* preserves the evidence, provenance, and version of each state transition. It turns the planned task and team structures into an explicit, queryable record of the run.

The conditions that a reliable state satisfies usually include *structured representation, governed updation, scoped visibility, and consistency management*. For structured representation, Magentic-One [84] externalizes shared execution progress through orchestrator-maintained Task and Progress Ledgers, while Graph of States [230] organizes structured belief states and constrains their transitions through causal graphs and state machines. Together, they illustrate a shift from private conversational context toward explicit and traceable runtime state. For governed updation, PatchBoard [467] validates agent-generated patches against schemas, role permissions, and runtime invariants before commitment, whereas MemTX [195] distinguishes tentative writes from transactional belief commits with explicit provenance and repair semantics. These mechanisms motivate an explicit proposal-validation-commit boundary between observed or proposed changes and authoritative state. For scoped visibility, Collaborative Memory [301] complements this process through identity- and time-scoped projections, showing that shared state can remain coordinated without requiring universal visibility. For consistency management, state generation, lost updates, and causal-order violations motivate isolation, causal ordering, and conflict-resolution mechanisms under concurrent writers [155]. Event-sourced designs provide a complementary mechanism by preserving committed transitions in append-only histories that support state reconstruction, replay, and branching [252].

Overall, these studies identify the basic requirements for reliable state recording, but they do not yet provide a unified graph-native implementation.

4.4.2 Fault Localization

Recording state does not by itself explain why an execution has gone wrong. In a long-horizon system, a local error may propagate through dependent tasks and agents, while the visible fault appears several steps after the original deviation. *Fault Localization* addresses this problem by detecting abnormal outcomes, locating the decisive error, tracing its effects through dependencies, and testing possible causes against available evidence. Runtime state supports this process by preserving dependencies, provenance, and evidence. The system treats the cause of a fault as a hypothesis and does not assume that temporal or structural links prove causality.

Structured state representations provide complementary mechanisms for localizing faults. Runtime state constrains reasoning from evidence to possible causes through explicit hypothesis-evidence dependencies and supports inspecting earlier states in detail and backtracking when evidence is insufficient [230], while MAGE [41] represents execution as paths in a hierarchical state tree, allowing erroneous branches and nearby valid decision boundaries to be identified. These approaches illustrate how structured execution state can constrain the search space for root-cause analysis. Failure-attribution studies further identify the information needed for this process: Who & When [465] attributes failures to both responsible agents and steps that caused the failure, MAST [28] distinguishes system-design, inter-agent coordination, and task-verification failures, and TraceElephant [35] considers execution traces, intermediate context, and complete inputs rather than final outputs alone. Together, they motivate preserving actors, transitions, dependencies, and validation evidence needed to formulate and test attribution hypotheses. Diagnosis ultimately requires validating such hypotheses against externally observable evidence. TDAD [4] connects code changes to affected tests through explicit code-test dependencies, while Cordon [42] uses typed lineage, shadow state, and semantic transaction boundaries to relate runtime actions to their external effects. These studies suggest that dependencies narrow the search for a cause, but they do not prove the cause. Recording the resulting diagnosis and its supporting evidence as part of runtime state then provides a traceable basis for subsequent recovery.

Overall, fault localization uses runtime records and external evidence to detect faults, trace their effects, test possible causes, and determine which parts of the execution remain valid for recovery.

4.4.3 Failure Recovery

Fault localization identifies where execution deviated, but system intelligence also needs a way to continue without discarding valid work or repeating harmful effects. *Failure Recovery* addresses this problem by selecting an explicit recovery boundary and determining how execution can safely resume from it. The system may retract invalid states, replay recoverable computation, compensate for external effects, or branch into an alternative execution path. The runtime state layer supports these operations by preserving committed versions, dependencies, provenance, and recovery boundaries, while distinguishing reconstructable internal states from external effects that require compensation.

Existing systems implement recovery mechanisms at different levels. MAGE [41], ALAS [93], CausalFlow [24], and ReflexGrad [147] localize failures and selectively repair affected execution regions, avoiding costly global recomputation. Event sourcing [252], AgentGit [196], and Shepherd [437] enable replay, rollback, and branching over recorded execution states, while DART [419] further constrains restoration to semantically valid boundaries under downstream dependencies and committed effects. Together, these approaches support localized recovery while preserving unaffected progress. In addition to the localized recovery mechanism mentioned, external effects require other recovery guidance that comes from the environment. SagaLLM [29] and RAC [279] combine checkpoints with compensation for effects that cannot be directly rolled back, while Atomix [249] coordinates reversible and irreversible effects through transactional settlement. Aegis [326] complements these mechanisms by improving agent-environment interactions to reduce environment-induced failures.

Overall, effective recovery requires selective repair with explicit handling of state dependencies and external effects. This process includes recording the recovery boundary, corrective actions, and resulting state, closing the recording, diagnosis, and recovery loop.

4.5 System Evolution

In open-ended and long-horizon environments, execution continuously generates evidence about effective structures, coordination strategies, and failure modes [365, 413, 420]. Task organization, agent coordination, and runtime state management provide the foundations for system intelligence, but do not inherently enable improvement over time.

To address this challenge, *System Evolution* leverages such experience to refine its organization and operation across executions. The evolution of the system level spans three dimensions: *task organization* improves objective decomposition and workflow construction; *agent coordination* adapts team structures and communication patterns; and *runtime state management* consolidates execution histories into reusable experience while enabling system updates to be validated, revised, or rolled back. These mechanisms turn runtime experience into sustained system-level improvement [268, 389, 453, 492].

Evolution of Task Organization. Predefined task structures are often inadequate in open-ended environments, where intermediate outcomes and changing conditions can invalidate prior decomposition and execution plans. Task evolution addresses this limitation by refining both task structures and workflows from execution feedback. At the task-structure level, TDAG [362] dynamically decomposes complex tasks and generates specialized agents as execution unfolds. Flow [256] refines subtask allocation using historical performance and prior workflow structures, while DynTaskMAS [435] dynamically maintains task dependencies to support adaptive scheduling and parallel execution. At the workflow level, DyFlow [363] determines subsequent operations from intermediate outputs and real-time feedback; EvoFlow [450] evolves heterogeneous workflow candidates through retrieval, crossover, mutation, and selection; and QualityFlow [127] uses intermediate quality checks to determine whether to proceed, clarify, or revert execution. Together, these approaches turn task organization from static planning into an iterative process in which execution outcomes refine subsequent decomposition and workflows. Such adaptability, however, also creates vulnerabilities: FlowSteer [181] shows that manipulated planning signals can steer replanning and dependency formation toward undesirable execution paths. Reliable task evolution therefore requires structural revisions to be grounded in trustworthy execution feedback.

Evolution of Agent Coordination. Changing task requirements and component capabilities can render predefined coordination structures ineffective. The evolution of agent coordination addresses this mismatch by adapting both *team structures* and *communication patterns* through collaborative experience. For *team structure evolution*, SwarmAgentic [471] jointly optimizes agent functionality and collaboration structures through feedback-guided population search. AgentNet [423] enables decentralized specialization and reorganization by adjusting agent connectivity and task routing according to local expertise and context, while self-organizing agents [60] show that specialized roles and shallow hierarchies can emerge without predefined assignments. Meta-Team [113] further leverages distributed execution experience to improve agent behavior, inter-agent coordination, and team organization across tasks. For *communication evolution*, DyTopo [228] reconstructs communication pathways at each reasoning round by matching agents' information needs and offerings. CARD [382] conditions communication structures on environmental changes in model capabilities, tools, and resources, while QueenBee Planner [335] distills execution traces and evaluation outcomes into reusable design rules for improving communication in subsequent tasks. Collectively, these approaches shift agent coordination from predefined collaboration toward experience-driven evolution of both team organization and information exchange.

Evolution of Runtime State Management. The evolution of runtime state management extends state from supporting execution and recovery to accumulating experience for future improvement. This involves two complementary processes: distilling execution histories into reusable knowledge and controlling state revisions to prevent erroneous experience from propagating. ReCreate [112] derives reusable domain patterns from interaction histories by analyzing the causes of success and failure. SkillGraph [254] distills failure cases into reasoning heuristics maintained in an evolving Skill Bank, while Swarm Skills [470] extracts successful trajectories into reusable coordination skills and refines them based on effectiveness, utilization, and freshness. Beyond experience accumulation, reliable evolution requires mechanisms for validating and revising persistent state. MemTX [195] separates tentative writes from validated belief commits and performs cascading repair when committed beliefs are retracted, limiting the propagation of invalid state. ActiveGraph [252] preserves event-sourced execution histories that support deterministic replay and efficient forking from prior states, enabling alternative branches to build on validated execution history. In summary, these studies make runtime state an experience substrate in which useful knowledge can be accumulated and reused, while unreliable updates can be revised, retracted, or bypassed.

Overall, system evolution enables system intelligence to improve across executions by turning runtime experience into system-level updates. Execution outcomes provide evidence for refining task organization, agent coordination, and state management, while validation and rollback mechanisms ensure that only reliable improvements persist. This establishes a closed loop between execution, experience, and evolution, allowing successful strategies to accumulate

and failures to inform subsequent decisions. System intelligence thus progresses from runtime adaptation toward sustained, experience-driven evolution.

5 Open Challenges and Research Opportunities

Graph Engineering provides a structural foundation for transforming LLM-based agent systems from individual intelligence into system intelligence by explicitly modeling work organization, component coordination, and state evolution. However, moving from task-specific graph structures toward general-purpose infrastructures that can operate continuously and be reused across systems introduces several unresolved challenges. These challenges concern not only how graphs are constructed, but also how they can evolve dynamically, operate reliably, scale efficiently, and interoperate across heterogeneous agent systems. More fundamentally, as the Work Organization Graph, Agent Team Graph, and State Evolution Graph become increasingly interconnected, the research focus must shift from merely constructing graph structures toward ensuring their consistent interpretation, reliable execution, and continual evolution.

5.1 Graph-Native Capability Substrates

Current Graph Engineering primarily makes the organization of tasks, agents, and runtime states explicit, while many capabilities used by an agent system are still maintained as independent collections or services [199]. Memory stores contain experiences and facts, skill libraries contain reusable procedures, and tool registries expose executable functions, but the relationships among these capabilities are often implicit [138, 348, 434]. As these repositories grow, capability selection becomes increasingly structural: a capability may depend on another capability, substitute for an unavailable one, compose with several others, require specific permissions, or be applicable only under particular runtime conditions.

Recent work on memory and skills already points toward graph-structured capability substrates [27, 74, 320]. A-MEM [405] dynamically links related memories into evolving memory networks, while Zep [296] represents changing facts and their temporal relations through a temporal knowledge graph. A similar transition is emerging for reusable skills. Graph of Skills [179] represents dependencies and workflow relations among skills to retrieve executable skill bundles rather than isolated entries, while SkillDAG [15] further allows typed skill relations to evolve from execution evidence. These systems suggest that graphs can organize not only tasks and agents, but also the capability substrate on which they operate.

A broader direction is therefore to construct unified *capability graphs* in which models, tools, skills, memories, data sources, verifiers, and execution environments are represented as typed nodes, with edges describing dependency, compatibility, composition, substitution, authorization, cost, and reliability. The main research challenge is not simply to represent each capability family as a graph, but to connect these capability graphs with task, agent, and runtime state graphs. Task decomposition should expose capability requirements, agent allocation should consider available capability subgraphs, and execution outcomes should update capability reliability and applicability. Such coupling would allow Graph Engineering to move from organizing system execution to organizing the reusable capability space from which execution is constructed [81, 186, 399, 469].

5.2 Self-Evolving Graph Systems

Existing Graph Engineering methods increasingly make graph structure an optimization variable. GPTSwarm [498] optimizes computational graphs of language agents, while workflow and topology optimization methods such as AFlow [456] and DyTopo [228] adapt executable or communication structures according to task feedback. Other approaches accumulate execution experience for future improvement: ReCreate [112] derives reusable patterns from successful and failed trajectories, while MemTX [195] and event-sourced agent designs [252] provide mechanisms for validating, revising, replaying, and forking persistent state. These developments represent early steps from fixed graph execution toward experience-driven structural adaptation.

However, runtime adaptation should be distinguished from persistent system evolution. Conditional routing, temporary worker assignment, or recovery may change one execution trajectory without changing the organization used in later tasks. A self-evolving graph system should instead transform execution evidence into persistent and reusable structural changes. This requires a closed process from execution and observation to structural credit assignment, graph modification, validation, and finally commit or rollback. Future systems must determine which task dependencies, agent

relations, capability assignments, or state structures were responsible for success and failure, and whether the resulting modification generalizes beyond the current execution.

An additional challenge is that these graphs cannot evolve independently. Modifying a task graph may change the capabilities required from the agent team, while replacing an agent may invalidate communication relations, permissions, or runtime assumptions. Future research should therefore study *cross-graph evolution*, where changes to task, agent, capability, and state graphs are coordinated under shared constraints. Structural evolution must also remain governable through provenance, versioning, validation, replay, and rollback. The long-term objective is not unrestricted self-modification, but systems that can accumulate useful organizational experience while preventing unreliable structural changes from propagating across executions.

5.3 Graph-Native Agent Operating Systems

The growing complexity of agent systems also raises an infrastructure question. Current engineering stacks separate model serving, harnesses, workflow engines, memory systems, multi-agent frameworks, and state stores, each using different abstractions for tasks, tools, messages, agents, events, and execution state. Protocols such as MCP [6] improve access to external capabilities, while graph-oriented frameworks such as LangGraph [166] provide explicit workflow and state representations. AIOS [239] takes a complementary operating-system view by providing scheduling, context, memory, storage, tool, and access-control services for LLM agents. However, these mechanisms do not yet provide a common structural substrate for organizing complete agent systems. AIOS itself is an important precedent: its kernel explicitly separates agent applications from scheduling, memory, storage, tools, and access control, showing why these concerns increasingly resemble operating-system services rather than application-specific logic [239].

A future *graph-native agent operating system* could make tasks, agents, capabilities, and runtime states first-class system objects represented through typed and versioned graphs. Instead of each framework separately implementing workflow scheduling, resource allocation, persistent state, communication, and recovery, a shared runtime could provide graph scheduling, capability discovery, state storage, event and provenance logging, structural transactions, permission enforcement, checkpointing, replay, rollback, and graph-level observability. Ontology Engineering would define the types, relations, and constraints of these objects, while the graph runtime would enforce their operational semantics.

Such an infrastructure would also provide the foundation required for safe system evolution. Execution traces could be linked directly to the graph structures that produced them, candidate structural changes could be evaluated against historical or counterfactual executions, and validated improvements could be committed as new graph versions. Graph Engineering would then evolve from a method for designing individual workflows or multi-agent topologies into a reusable system substrate for constructing, executing, observing, and continuously improving agent systems. This progression from shared semantics, to graph-structured capabilities, to controlled structural evolution, and finally to graph-native runtime infrastructure represents a possible path toward scalable and persistent System Intelligence.

5.4 Privacy and Ethics

System intelligence introduces broader privacy and ethical risks because it coordinates multiple agents, tools, memories, and shared states over long horizons. Compared with a single-agent setting, sensitive information may be replicated across components, propagated through workflows, and preserved in persistent state, increasing the risk of unauthorized access, cross-task leakage, and unintended inference of private attributes from execution traces. Moreover, as decisions are distributed across interacting components, accountability becomes harder to assign when biased evidence, faulty reasoning, or adversarial inputs are amplified through the system. Future system-intelligent agents therefore require privacy-preserving state management, scoped permissions, provenance-aware logging, and strong human oversight to ensure that autonomy does not come at the cost of user privacy, fairness, or controllability.

6 Future Direction: Ontology Engineering for Next-Generation System Intelligence

Graph Engineering provides a structural foundation for system intelligence by making relationships among work organization, agent coordination, and runtime state explicit, schedulable, and adaptable. However, explicit graph structures alone do not ensure that system entities and relations are defined consistently. Many existing approaches assume that goals, operations, agent capabilities, and runtime states already have clear and shared meanings. This assumption often

fails in open, long-running, and industrial environments, where the same concept may be defined differently across graph views, system components, or stages of execution. Ontology Engineering [71, 103, 204] addresses this limitation by establishing a shared, machine-interpretable model of system entities, relations, and constraints. It therefore provides the semantic foundation needed to connect, validate, reuse, and evolve graph structures, supporting the next generation of system intelligence.

6.1 Limitation of Graph Engineering-based System Intelligence

End-task success alone is insufficient to determine whether a system has developed System Intelligence. Performance gains may result from a stronger foundation model, longer context, additional reasoning samples, or greater computational cost rather than more effective task organization, agent coordination, or state management. Future evaluation should distinguish component-level capability from the contribution of system organization. It should assess goal formation, semantic consistency, parallel execution efficiency, heterogeneous capability allocation, collective decision quality, state consistency, failure recovery, transfer across tasks, and runtime overhead. Evaluation tasks should also include incomplete objectives, concurrent workloads, distributed information, component failures, and environmental changes to test whether the system can maintain coherent behavior under structural disturbances. Beyond end-to-end metrics, intervention studies, structural ablations, and execution-trace analysis are needed to identify the causal contributions of different system mechanisms and distinguish genuine system-level capability from gains produced by additional computation.

These challenges clarify the limits of Graph Engineering. Graph structures can explicitly organize relationships among tasks, components, and runtime states, but System Intelligence must also formulate appropriate goals, establish a shared and grounded understanding of the system, and support rigorous system-level evaluation. Ontology Engineering primarily addresses the need for shared semantics and can also provide consistent definitions for goals, roles, states, evidence, and operational constraints. It should therefore be viewed as a semantic foundation connecting Graph Engineering to broader System Intelligence rather than a complete solution to all system-level challenges.

Graph Engineering makes relationships among tasks, agents, and runtime states explicit, but explicit structures do not ensure that system components interpret them consistently. Agents may still disagree about what constitutes task completion, sufficient evidence, valid state, or authorized action. Ontology Engineering addresses this broader problem by establishing a shared, machine-interpretable model of the system. Rather than merely adding semantic annotations to graphs, it defines which entities exist, what their relations mean, which constraints must hold, and what conclusions can be derived from them.

An ontology for System Intelligence should be layered and modular. A core ontology can define concepts shared across systems, while specialized modules describe goals and values, agents and capabilities, observations and evidence, actions and states, and evaluation criteria. Domain ontologies can further extend these concepts for particular applications. Such a structure provides consistent definitions of Goals, Agents, Capabilities, Evidence, Policies, States, and Outcomes without requiring every system or domain to adopt a single monolithic model.

6.2 Goal Formation and Value Alignment

For Goal Formation and Value Alignment, Ontology Engineering can represent the provenance, priority, authorization scope, completion criteria, and constraints of candidate goals. These representations allow a system to identify goal conflicts, detect unauthorized modifications, and determine what evidence is required for completion. Ontologies cannot decide which values a system should adopt, but they can make goals and normative constraints explicit and verifiable. Recent ontology-guided agent systems illustrate this shift from representing domain concepts to constraining agent reasoning [266, 309, 364]. In LAMP, a Planner, Builder, and Verifier collaboratively access a domain-specific ontology through MCP, using explicit structured knowledge at inference time rather than relying solely on model parameters [7]. Likewise, Agentology proposes treating the ontology-defined environment, rather than the individual agent prompt, as the primary object of system design, allowing multiple specialist agents to reason over a shared and persistent semantic structure [266]. These developments point toward an ontology-centered organization of multi-agent systems in which semantic constraints are externalized from individual agents and shared across the system.

6.3 Shared Semantics and World Grounding

For Shared Semantics and World Grounding, ontologies provide common definitions and mappings across agents and systems. These concepts must also be connected to tool outputs, environmental observations, timestamps, provenance, and validation results, since semantic consistency alone does not guarantee factual correctness. Recent multi-agent ontology-enrichment systems demonstrate how this semantic layer can itself be dynamically maintained. OntoCodex coordinates decision, ontology-reading, knowledge-base, terminology, and script-generation agents to enrich an existing OWL ontology while preserving its structural constraints and grounding newly introduced concepts in curated knowledge sources [77]. CoA-Text2OWL similarly distributes ontology learning across multiple worker agents and a manager agent, demonstrating the potential of agent collaboration for constructing coherent ontologies from large textual sources [10]. Ontology-grounded tool and agent designs, such as AgentO and Ontology-to-Tools, further connect semantic concepts to executable capabilities and tool interfaces [71, 490]. These systems indicate that future ontology infrastructure may not remain static: agents can participate in proposing, validating, aligning, and updating the semantic model while retaining explicit provenance and human oversight.

6.4 Measuring System Intelligence

Ontology Engineering can also support the measurement of System Intelligence by standardizing the meanings of task success, failure, agent contribution, recovery, state consistency, and runtime cost. Shared representations of system configurations, execution events, evidence, interventions, and outcomes would make execution traces more comparable across systems and support structural ablation and causal analysis. Ontologies do not replace evaluation methods, but they clarify what is being measured and whether a metric concerns foundation-model capability, individual-agent performance, or system-level organization [184, 270]. By providing a common vocabulary for system-level events and entities, ontology specifications such as Ontology SLR and Palantir Ontology could facilitate more systematic comparison of heterogeneous agent architectures [184, 270].

Future research should investigate how system ontologies are grounded, updated, and governed. LLMs may assist in proposing new concepts and relations, but semantic changes should undergo provenance checking, consistency validation, and impact analysis [77, 204, 266, 490]. System ontologies must also support version control, compatibility checking, migration, and rollback as tasks and environments change. Their constraints should be connected to runtime mechanisms that enforce permission checks, evidence requirements, and valid state transitions. Recent work such as LAMP demonstrates one direction in which structured ontology knowledge is directly exposed to agents through tool interfaces, while Agentology explicitly treats the ontology as part of the operational environment within which multiple agents reason [266, 490]. Ontology Engineering thus defines the shared conceptual model of System Intelligence, Graph Engineering instantiates this model as task-specific structures, and runtime mechanisms enforce its operational consequences.

7 Benchmarks, Datasets, and Evaluation

Evaluation should follow the unit of intelligence being studied. Model Intelligence concerns capabilities expressed within bounded model interactions. Individual Intelligence concerns whether a single autonomous agent can combine reasoning with external capabilities and environmental feedback over a sustained trajectory. System Intelligence further concerns whether multiple intelligent components and their relations can be organized, coordinated, maintained, and improved as a coherent system. We therefore organize evaluation resources around these three levels rather than by task domain or graph type.

We distinguish three forms of evaluation resource. A *benchmark* defines tasks, an evaluation protocol, and scoring rules. A *dataset* provides reusable instances, annotations, graphs, interaction records, or execution traces. An *environment* exposes executable state that an agent or agent system can observe and modify. These forms are not mutually exclusive. Table 1 uses B, D, and E to denote benchmark, dataset, and executable environment, respectively.

7.1 Model Intelligence

Model Intelligence evaluation focuses on capabilities expressed within bounded model interactions. Representative benchmarks cover broad knowledge and reasoning, instruction following, executable code generation, multimodal understanding, and retrieval-augmented reasoning [33, 117, 211, 298, 360, 388, 444, 487]. Recent resources also

Table 1 : Representative benchmarks, datasets, and executable environments across Model, Individual, and System Intelligence. Type: B = benchmark or evaluation protocol; D = released dataset, annotations, or traces; E = executable or interactive environment. Focus denotes the principal capability or structural property evaluated.

Name	Type	Primary Unit	Focus	Evaluation	Link
			Model Intelligence		
MMLU/MMLU-Pro [117, 360]	B/D	QA instances	Knowledge	Broad knowledge and problem solving; MMLU-Pro increases reasoning difficulty and prompt robustness.	
GPQA [298]	B/D	Expert science QA	Reasoning	Graduate-level scientific knowledge and difficult multi-step reasoning.	
NPPC [412]	B/D	NP instances	Reasoning	Scalable, automatically verifiable reasoning over NP-complete problems.	
OlymMATH [330]	B/D	Math problems	Reasoning	Olympiad-level reasoning with objective and formal verification.	
IFEval [487]	B/D	Instruction pairs	Following	Verifiable instruction following under objectively checkable constraints.	
EvolIF [139]	B/D	Multi-turn dialogues	Following	Evolving instruction following, constraint tracking, and failure recovery.	
HumanEval/EvalPlus [33, 211]	B/D	Coding problems	Coding	Executable functional correctness with strengthened test coverage.	
MMMU [444]	B/D	Multimodal QA	Multimodal	Expert-level multimodal understanding and reasoning across disciplines.	
OMHBench [157]	B/D	Omni-modal QA	Multimodal	Grounded multi-hop reasoning across text, vision, and speech.	
LiveBench [375]	B/D	Refreshable tasks	General	Frequently refreshed capability evaluation with objective scoring to reduce contamination.	
GraphRAG-Bench [388]	B/D	RAG tasks	Retrieval	Graph construction, retrieval, reasoning, and generation in GraphRAG.	
			Individual Intelligence		
A^2E [345]	B/D	Harness executions	Harness	End-to-end agent harness auditing; execution efficiency, tool use, task planning, and error recovery.	
AgentBench [216]	B/D/E	Agent trajectories	General	Reasoning and decision making across multiple interactive environments.	
GAIA [243]	B/D	Assistant tasks	General	Integrated reasoning, browsing, multimodal understanding, and tool use.	
AgencyBench [185]	B/D/E	Long-horizon tasks	General	Long-horizon real-world autonomy with tools and extended context.	
WebArena [488]	B/D/E	Web trajectories	Web	Long-horizon interaction with realistic websites and execution-based evaluation.	
OSWorld [394]	B/D/E	Computer trajectories	Computer	Open-ended interaction with real desktop applications and operating systems.	
SWE-bench [144]	B/D/E	Repository tasks	Software	Repository-level software issue resolution with executable verification.	
AppWorld [338]	B/D/E	Application state	Tools	API use, code generation, application-state transitions, and task completion.	
τ-bench [429]	B/D/E	Agent-tool dialogues	Tools	Policy following, user interaction, tool execution, final state, and repeated-run reliability.	
ToolSandbox [227]	B/D/E	Stateful dialogues	Tools	Stateful tool execution, dependencies, intermediate milestones, and recovery.	
AgentDojo [55]	B/D/E	Adversarial episodes	Security	Agent utility and robustness under prompt injection attacks.	
Harness-Bench [430]	B/D/E	Sandboxed workflows	Harness	Effects of context, tools, state, constraints, permissions, tracing, and recovery.	
Skill-Use [111]	B/D/E	Skill tasks	Skills	Skill triggering, procedural compliance, capability boundaries, and harness dependence.	
LongMemEval [380]	B/D	Multi-session QA	Memory	Information extraction, temporal reasoning, updates, and long-term interactive memory.	
MemoryAgentBench [128]	B/D	Multi-turn memory	Memory	Retrieval, test-time learning, long-range understanding, and selective forgetting.	
MemoryArena [116]	B/D/E	Multi-session tasks	Memory	Acquisition and reuse of experience across interdependent sessions.	
GateMem [299]	B/D	Memory episodes	Memory	Access control, deletion, selective forgetting, and memory governance.	
MemSyco-Bench [387]	B/D	Memory decisions	Memory	Appropriate use of retrieved memory under factual, scope, conflict, update, and personalization conditions.	
Mem2ActBench [319]	B/D	Memory-tool chains	Memory	Contribution of retained memory to subsequent tool actions.	
LongDS-Bench [404]	B/D/E	Long trajectories	Long-term	State maintenance, restoration, adaptation, and rollback over long executions.	
EvoMemBench [366]	B/D	Memory episodes	Evolution	Memory evolution and selective retention within and across episodes.	
Trainee-Bench [86]	B/D/E	Workplace streams	Evolution	Scheduling, exploration, and continual learning in dynamic workplaces.	
SEA-Eval [142]	B/D/E	Task streams	Evolution	Cross-task evolutionary gain, stability, and execution efficiency.	
Evo-Bench [133]	B/D	Harness evolution	Evolution	Autonomous harness improvement and cross-domain transfer.	
OpenClawBench [219]	B/D	Execution traces	Failure	Process-side anomalies, robustness, and failures in real agent trajectories.	
BenchTrace [130]	B/D	Repeated episodes	Reflection	Whether reflection on failures improves behavior in subsequent executions.	
TheAgentCompany [402]	B/D/E	Workplace episodes	Long-term	Long-horizon workplace tasks spanning browsing, coding, and communication.	
			System Intelligence		
TaskBench [318]	B/D	Tool graphs	Work	Task decomposition, tool selection, grounding, and explicit tool-graph construction.	
WorFBench [289]	B/D	Workflow graphs	Work	Workflow generation with sequence-level and graph-level structure matching.	
FlowBench [391]	B/D	Workflow pairs	Work	Workflow-guided planning across heterogeneous workflow representations.	
ComfyBench [408]	B/D/E	Executable workflows	Work	Construction and execution of explicit node-edge workflows.	
TPS-Bench [403]	B/D/E	Scheduling tasks	Work	Dependency-aware planning, parallel scheduling, throughput, and execution efficiency.	
JourneyBench [18]	B/D/E	Policy workflows	Work	Policy-constrained service workflows and business-rule adherence.	
ETOM [65]	B/D/E	Tool hierarchies	Work	Hierarchical orchestration, server selection, and out-of-scope robustness.	
LLM-Coordination [1]	B/D/E	Coordination games	Team	Joint planning, theory of mind, sustained coordination, and partner robustness.	
VillagerBench [66]	B/D/E	Minecraft tasks	Team	Workload distribution, task dependencies, adaptation, and synchronized execution.	
MultiAgentBench [494]	B/D/E	Multi-agent episodes	Team	Collaboration, competition, milestones, and topology-sensitive coordination.	
AgentsNet [102]	B/D/E	Networked tasks	Evolution	Self-organization, adaptive communication, and network scaling.	
DBS [341]	B/D	Workflow graphs	Evolution	Adaptive workflow synthesis under distributed heterogeneity and privacy.	
SILO-BENCH [472]	B/D/E	Distributed tasks	Team	Role-free coordination under information silos and agent scaling.	
CoLLAB [237]	B/D	Coordination tasks	Team	Constraint-based coordination and structural credit assignment.	
Collab-Overcooked [329]	B/D/E	Collaboration games	Team	Process-oriented collaboration quality beyond final task success.	
MAS-BENCH [421]	B/D/E	Distributed sorting	Team	Shared-state consistency, protocol alignment, termination, and agent scaling.	
DPBench [114]	B/D/E	Contention games	Team	Sequential and simultaneous coordination under shared-resource contention.	
CalBench [499]	B/D/E	Decentralized tasks	Team	Coordination, communication efficiency, fairness, and privacy under private information.	
TAMAS [153]	B/D/E	Adversarial episodes	Team	Robustness and safety under adversarial multi-agent interaction.	
SyncBench [110]	B/D/E	Recovery instances	State	Belief-world consistency, diagnosis, resource awareness, and recovery.	
MAST [28]	D	Failure annotations	State	Multi-agent failure modes in system design, alignment, and verification.	
Who&When [465]	B/D	Failed trajectories	State	Attribution of failures to responsible agents and decisive execution steps.	
Who&When Pro [212]	B/D	Failed trajectories	State	Large-scale responsibility and temporal attribution under controlled failures.	
TraceElephant [35]	B/D/E	Execution traces	State	Failure attribution under complete execution observability.	
MP-Bench [135]	B/D	Failure cases	State	Multi-perspective evaluation when failures admit several plausible attributions.	
R2Act [283]	B/D/E	Incident states	State	Diagnosis-to-action reasoning, admissible recovery, and recovery validity.	
MAFBench [265]	B/E	Framework runs	Evolution	Framework-level comparison of orchestration, planning, coordination, and scalability.	
MASEval [72]	B/E	System variants	Evolution	Topology, orchestration, framework, and runtime design comparison.	
MAS-PromptBench [14]	B/E	MAS configurations	Evolution	Optimization across workflow topologies, protocols, and team sizes.	
BenchAgent [88]	B/E	Agent workflows	Evolution	Controlled comparison of single, fixed multi-agent, and evolving agent workflows.	

address the rapid saturation of static evaluation: LiveBench refreshes questions to reduce contamination [375], while NPPC generates automatically verifiable NP-complete problem instances with scalable difficulty [412]. The primary evaluation unit at this level remains the model output; persistent interaction and environment state are largely outside the evaluation target.

7.2 Individual Intelligence

Individual Intelligence shifts the evaluation unit from outputs to trajectories. AgentBench and GAIA evaluate general agent capabilities, while WebArena, OSWorld, SWE-bench, AppWorld, and TheAgentCompany test sustained interaction with web, computer, software, API, and workplace environments [144, 216, 243, 338, 394, 402, 488]. More recent benchmarks push this setting toward longer and less idealized execution: AgencyBench evaluates extended real-world tasks [185], AgentGym2 introduces tool discovery and robustness to noisy and underspecified information [385], and LongCLI-Bench targets long-horizon command-line software engineering [82]. Tool and Harness resources further evaluate whether an agent can reliably access and govern external capabilities [55, 111, 227, 429, 430].

Long-horizon evaluation additionally examines whether information and experience remain useful across extended or repeated executions. Existing resources cover long-term memory, memory governance, memory-to-action transfer, and reliable use of retrieved memories [21, 116, 128, 299, 319, 380, 387]. Recent benchmarks increasingly move beyond isolated episodes toward explicit adaptation and evolution: A^2E provides an end-to-end evaluation engine for agent harnesses, capturing standardized execution traces and assessing harness capabilities in execution efficiency, tool use, task planning, and error recovery [345]; Trainee-Bench evaluates scheduling, exploration, and continual learning in dynamic workplace streams [86]; SEA-Eval measures evolutionary gain and stability across sequential tasks [142]; and Evo-Bench evaluates whether models can improve their own agent harnesses [133]. Other resources examine persistent state, evolving memory, process anomalies, and reflection across executions [130, 219, 366, 404]. These benchmarks evaluate increasingly persistent and adaptive agents, but responsibility for task organization and execution remains centered on one agent or one local runtime.

7.3 System Intelligence

System Intelligence expands evaluation from an individual trajectory to the organization of multiple components and their relations. Existing resources provide partial probes of this broader objective. Work-oriented benchmarks evaluate decomposition, workflow structure, dependency-aware scheduling, and hierarchical orchestration [18, 65, 289, 318, 391, 403, 408]. Coordination benchmarks evaluate collaboration, communication topology, distributed information, resource contention, scalability, privacy, and adversarial robustness [1, 66, 114, 153, 237, 329, 421, 472, 494, 499]. State-oriented resources further expose failure attribution, consistency, diagnosis, and recovery as explicit system-level evaluation targets [28, 35, 110, 135, 213, 283, 465].

Evaluation of system adaptation and evolution is also beginning to emerge. AgentsNet examines self-organization and scaling of networked agents [102], while DBS studies adaptive workflow synthesis under distributed heterogeneity and privacy [341]. MASEval treats topology, orchestration, framework, and runtime design as system-level evaluation variables [72], and MAS-PromptBench evaluates optimization across different multi-agent configurations [14]. MAF-Bench further compares alternative agent-framework designs [265], while BenchAgent directly contrasts single-agent, fixed multi-agent, and evolving workflows under controlled protocols [88]. Despite this progress, persistent system evolution remains comparatively underexplored: current resources rarely evaluate whether runtime evidence produces durable and transferable improvements to work organization, team structure, and runtime management across repeated executions.

7.4 Evaluation Principles and Open Challenges

Across all three levels, evaluation should report *effectiveness*, *efficiency*, and *robustness*. Graph-engineered systems additionally require *structural fidelity*, *operational correctness*, and *evolution and governance*. These dimensions distinguish whether a system succeeds from whether its underlying structure is valid, its graph operations are executed correctly, and its structural changes remain traceable and controllable.

Three gaps are especially important. First, system-level improvements must be separated from gains caused by stronger models, larger contexts, additional tools, retries, or compute. Second, current resources remain fragmented across work organization, coordination, runtime state, and evolution, making cross-structure effects difficult to measure. Third, structural credit assignment and dynamic system-level evaluation remain weak. Future benchmarks should therefore provide matched execution budgets, versioned graph artifacts, complete traces and state snapshots, controlled structural perturbations, and repeated evaluations across tasks and time.

8 Open-Source Libraries and Engineering Ecosystem

Open-source libraries translate the evolution from Model Intelligence to Individual Intelligence and System Intelligence into executable engineering stacks. Modern libraries often span several levels: a model-serving engine may also serve as the rollout backend of reinforcement learning, while an agent framework may support both a single tool-using agent and a multi-agent workflow. We therefore organize libraries by their *primary engineering target* rather than by exclusive functionality. Model Intelligence libraries primarily construct, post-train, or execute model capabilities; Individual Intelligence libraries provide the persistent capabilities and control required by an autonomous agent; and System Intelligence libraries organize multiple intelligent components, their relations, shared execution structures, and runtime state.

We include reusable projects whose source and technical documentation are publicly available and whose abstractions directly affect the construction or execution of intelligent systems. Table 2 summarizes representative systems. The *Focus* column records their main engineering concerns but is intentionally non-exclusive. Generic machine learning utilities, graph databases, workflow schedulers, and domain-specific agent applications are omitted unless they expose a reusable abstraction that is directly relevant to the intelligence stack. Source-available systems are retained when they have substantial engineering relevance, but their licensing status is stated explicitly.

8.1 Model Intelligence

At the Model Intelligence level, open-source infrastructure determines how model capabilities are constructed, refined, and exposed to higher layers. Transformers provides a common model-definition and execution interface, while Megatron Core addresses large-scale distributed pretraining [258, 323, 377]. LLaMA-Factory packages supervised and preference-oriented post-training into a unified toolkit, whereas verl and slime focus on scalable reinforcement learning pipelines that connect training, rollout generation, reward computation, and increasingly agentic environment interaction [321, 480, 496]. vLLM and SGLang provide the inference and rollout substrate on which both interactive agents and modern post-training systems depend [163, 479]. These libraries primarily engineer model parameters and model execution rather than persistent agent behavior or system organization.

8.2 Individual Intelligence

At the Individual Intelligence level, the main engineering object shifts from model parameters to the runtime surrounding a model. LangChain, OpenAI Agents SDK, Claude Agent SDK, and Pydantic AI expose variants of the model–tool loop together with middleware, permissions, validation, sessions, state, and human control [11, 165, 262, 282]. LlamaIndex Workflows, Haystack, and Burr provide more explicit control over context construction, workflow transitions, event routing, and persistent execution [12, 58, 223]. Their abstractions closely match the progression from Context Engineering to Harness Engineering and Loop Engineering: external capabilities are made accessible to the model and then organized into persistent, observable execution processes.

Persistent information is increasingly treated as another runtime capability. Letta maintains long-lived agent state and memory, while Graphiti represents changing contextual knowledge as a temporal graph with provenance [175, 296]. MCP addresses a complementary problem by standardizing how tools, resources, and prompts are exposed across agent runtimes [248]. Langflow and Dify lower the implementation barrier through visual workflow composition [167, 168]. Several of these systems can also compose multiple agents, but their primary abstractions remain centered on building and operating an agent or agent application rather than explicitly engineering system-level organization.

8.3 System Intelligence

System Intelligence libraries make relationships among tasks, agents, executors, and shared state explicit engineering objects. LangGraph, Microsoft Agent Framework, and Google ADK provide graph-oriented execution models in which agents and deterministic operations can be composed through conditional, concurrent, cyclic, or collaborative structures [98, 166, 245]. AutoGen established an influential multi-agent programming model based on message-passing agents and flexible conversation patterns, although it is now maintained primarily for existing users [244, 381]. AG2 represents a community continuation of this lineage with its own agent protocol and multi-agent abstractions [342].

Other systems place stronger emphasis on organizational semantics. CrewAI separates role-oriented Crews from event-driven Flows, while CAMEL's Workforce couples task decomposition with worker assignment and hierarchical co-

Table 2 : Representative open-source projects and engineering systems across Model, Individual, and System Intelligence. Libraries are grouped by their primary engineering target rather than exclusive functionality. Focus summarizes the main engineering concerns exposed by each system.

Project/System	Focus	Abstraction	Engineering Paradigm	License	Link
			Model Intelligence		
Transformers [377]	Model interface	Unified model definitions, configurations, tokenizers, and generation APIs	Model loading, training, generation, multimodal models, and integration with downstream training and inference stacks	Python; Apache-2.0	
Megatron Core [258, 323]	Pretraining	Distributed transformer training building blocks	Tensor, pipeline, data, expert, and context parallelism; mixed precision and scalable distributed training	Python; Apache-2.0	
LLaMA-Factory [480]	Post-training	Unified fine-tuning and post-training recipes	Continued pretraining, SFT, preference optimization, reward modeling, PPO, LoRA, and quantized fine-tuning	Python; Apache-2.0	
verl [321]	RL post-train	Distributed RL post-training dataflow	PPO, GRPO and related algorithms; integration with FSDP/Megatron for training and vLLM/SGLang for rollout generation	Python; Apache-2.0	
slime [496]	RL scaling	Training–rollout–data-buffer loop	Megatron training, SGLang rollout, custom rewards, verifiers, tool interaction, sandboxes, and asynchronous agentic data generation	Python; Apache-2.0	
vLLM [163]	Serving	PagedAttention-based inference engine	High-throughput batched inference, continuous serving, efficient KV-cache management, and model-serving APIs	Python/CUDA; Apache-2.0	
SGLang [479]	Serving/rollout	Structured generation frontend and high-performance runtime	Prefix-cache-aware execution, structured outputs, parallel inference, distributed serving, and rollout integration	Python/CUDA; Apache-2.0	
			Individual Intelligence		
LangChain [165]	Harness/Loop	Agent loop over models, tools, middleware, and state	Dynamic tools, middleware, tool retries, context control, structured output, state persistence, and human intervention	Python; MIT	
OpenAI Agents SDK [262]	Harness/Loop/Team	Agent runner with tools, guardrails, sessions, and delegation	Tool execution, agents-as-tools, handoffs, guardrails, sessions, HITL, tracing, and multi-agent composition	Python; MIT	
Claude Agent SDK [11]	Harness/Exec.	Programmable Claude Code agent runtime	Filesystem and shell tools, permission control, MCP tools, hooks, sessions, custom tools, and programmatic subagents	Python; MIT	
Pydantic AI [282]	Harness/State	Typed agents, capabilities, and `pydantic-graph`	Typed tools and outputs, validation, MCP, HITL approval, graph/state-machine control, and durable execution integrations	Python; MIT	
LlamaIndex Workflows [223]	Context/Workflow	Event-driven asynchronous workflow of typed steps and events	Retrieval-oriented agents, event routing, branching, loops, parallel steps, persistence, recovery, and HITL	Python; MIT	
Haystack [58]	Context/Workflow	Modular pipelines and agent workflows	Retrieval, routing, memory, tools, conditional branches, loops, component composition, tracing, and deployment	Python; Apache-2.0	
Apache Burr [12]	Loop/State	Action graph interpreted as a persistent state machine	Explicit transitions and state updates, persistence, resumability, streaming, HITL, telemetry, and trace inspection	Python; Apache-2.0	
Letta Agent SDK [175]	Memory/State	Stateful agent backed by a persistent agent harness	Persistent memory, sessions, skills, subagents, local or remote execution, and long-lived personalized agent state	TypeScript; Apache-2.0	
Graphiti [296]	Memory/Graph	Temporal context graph of entities, episodes, facts, and provenance	Incremental graph updates, temporal validity, changing facts, source provenance, ontology support, and historical retrieval	Python; Apache-2.0	
MCP Python SDK [248]	Capability I/O	Standard client/server interface for resources, tools, and prompts	Capability discovery, tool execution, context resources, prompts, lifecycle management, authentication, and interoperable transports	Python; MIT	
Langflow [167]	Visual workflow	Visual node-edge canvas for agents, models, tools, and data	Visual composition, reusable components, agent/tool integration, MCP exposure, execution inspection, and deployable flows	Python/TS; MIT	
Dify[b] [168]	Visual workflow	Visual Workflow/Chatflow graph and application runtime	RAG, agents, tools, branching, loops, variables, triggers, HITL, node-level traces, and workflow versions	Python/TS; source-available	
			System Intelligence		
LangGraph [166]	Work/Team/State	Typed `StateGraph` of nodes, edges, reducers, and subgraphs	Conditional and cyclic routing, parallel fan-out, multi-agent composition, durable execution, checkpoints, interrupts, replay, and state inspection	Python/TS; MIT	
Microsoft Agent Framework [245]	Work/Team/State	Graph-based workflows of agents and deterministic executors	Sequential, concurrent, handoff, and group collaboration; checkpoints, time travel, HITL, middleware, streaming, and tracing	Python/.NET; MIT	
Google ADK [98]	Work/Team/State	Workflow graphs combining agents and executable nodes	Sequential, parallel, loop, graph, dynamic, and collaborative workflows; routing, session state, evaluation, and deployment	Python; Apache-2.0	
AutoGen/GraphFlow[a] [244, 381]	Work/Team	Event-driven agents and explicit multi-agent interaction patterns	Message passing, group chat, distributed runtime, tool execution, GraphFlow-style directed interaction, logging, and inspection	Python; MIT	
AG2 [342]	Team/Harness	Protocol-driven agents and multi-agent orchestration	Tools, HITL, agent cooperation, multi-agent conversation patterns, knowledge, compaction, and extensible agent protocols	Python; Apache-2.0/MIT	
CrewAI [51]	Work/Team/State	Role/task-based Crews plus event-driven Flows	Role specialization, task ownership, sequential and hierarchical processes, event routing, shared state, persistence, callbacks, and tracing	Python; MIT	
CAMEL [182]	Work/Team/Evol.	Workforce hierarchy and task-dependency structure	Task decomposition, capability-based assignment, parallel workers, dependencies, role interaction, failure handling, shared memory, and workforce state	Python; Apache-2.0	
Mastra[c] [238]	Work/Team/State	Agents plus graph-based workflow engine	Sequential, branch, and parallel flows; agent composition, memory, HITL, suspend/resume, storage-backed state, MCP, evaluation, and observability	TypeScript; Apache core	
GPTSwarm [498]	Team/Evolution	Optimizable computational graph of LLM operations and agents	Agent-graph construction, composite swarm graphs, node and prompt optimization, inter-agent edge creation or pruning, cost tracking, and graph optimization	Python; MIT	

[a] AutoGen is in maintenance mode and is retained because of its historical influence on multi-agent programming; Microsoft recommends Agent Framework for new projects. [b] Dify uses a modified Apache-2.0 license with additional deployment and branding restrictions and is included as a source-available ecosystem reference. [c] Mastra's core is Apache-2.0, while code under its enterprise directories is governed by a separate enterprise license.

ordination [51, 182]. Mastra combines agents with an explicit workflow engine and persistent execution state [238]. GPTSwarm is particularly relevant to Graph Engineering because it treats graph connectivity itself as an optimization variable: both node-level prompts and inter-agent edges can be modified to improve the resulting system [498]. Nevertheless, most production-oriented frameworks still operate within developer-defined organizational templates. Dynamic routing is common, but persistent creation, removal, or rewiring of system structure from accumulated execution evidence remains rare.

8.4 Open Challenges in the Engineering Ecosystem

The ecosystem shows a clear progression from model infrastructure to persistent agent runtimes and multi-component orchestration, but the boundaries between these layers remain fragmented. Model training and serving systems expose different execution semantics from agent runtimes; agent frameworks use incompatible representations of tools,

messages, workflows, events, and state; and multi-agent systems rarely share a common representation of task dependencies, capabilities, authority, communication, and runtime state. Protocols such as MCP improve capability interoperability, but they do not provide a common representation for executable system organization.

A second limitation is that current dynamism is primarily *within* predefined structures. Conditional edges, routing, parallel fan-out, worker assignment, and recovery can change an execution path without changing the persistent organization that governs future executions. GPTSwarm and a small number of research-oriented systems expose topology optimization, but systematic cross-run evolution remains uncommon. This creates a gap between current orchestration frameworks and the RSI view of Graph Engineering, where runtime evidence should be abstracted into reusable structural changes.

Finally, state remains divided among model checkpoints, agent memories, workflow snapshots, message histories, event logs, and temporal knowledge stores. Existing observability tools can reconstruct what executed, but they seldom capture typed causal relations between observations, decisions, structural mutations, failures, recovery actions, and later system improvements. A more complete Graph Engineering substrate should therefore support typed and versioned work, team, and runtime structures; safe structural transactions and validators; persistent provenance; replay and rollback; graph-level tracing and counterfactual comparison; and controlled mechanisms for retaining successful structural changes across executions.

9 Applications of Graph Engineering

Applications provide a complementary view of Graph Engineering. Unlike benchmarks and open-source libraries, which can be organized naturally by intelligence level, applications are better distinguished by the domains in which structural decisions affect real work. We therefore organize this section by application domain while using intelligence level and Graph Engineering focus as cross-domain descriptors.

We include both research prototypes and deployed agent systems when task organization, agent relations, or runtime state have operational consequences. The term *Graph Engineering* need not be used explicitly by the original system. A workflow, team, dependency structure, or persistent environment is relevant when changing that structure changes how the system executes. Systems that use a knowledge graph only as an external retrieval source are not included unless the graph also affects task organization, agent coordination, or runtime behavior.

Table 3 uses I and S to denote Individual and System Intelligence. Some systems span both levels as they evolve from a single persistent agent toward parallel or multi-agent execution. The *Focus* column maps each application to the current Graph Engineering directions of Work Organization, Agent Team, Runtime State, and System Evolution. These assignments are non-exclusive and reflect our interpretation of the operational structure exposed by each system. Pure Model Intelligence applications are omitted because model capability alone does not constitute Graph Engineering without persistent agent execution or system structure.

9.1 Software Engineering and IT Operations

Software engineering is one of the clearest domains in which the progression from Individual to System Intelligence is already visible. Early multi-agent systems such as MetaGPT and ChatDev structured software development around predefined stages and specialist roles [119, 286], while SWE-agent showed that the interface between an agent and a repository can itself strongly shape execution [418]. OpenHands broadened this interaction model through a persistent event stream connecting code, shell, browser, and delegation [359].

Recent coding systems increasingly make parallel agent work an explicit engineering object. Codex supports concurrent agents operating in isolated worktrees, while Claude Code combines subagents, checkpoints, hooks, background execution, and agent teams [10, 261]. OpenCode exposes configurable primary agents and subagents, whereas Cline represents tasks and dependencies on a shared board and persists team state across sessions [5, 50]. These systems shift software engineering from managing one agent trajectory toward managing concurrent work, isolated branches, dependencies, test feedback, and merge decisions. Project ALICE extends the same structural view to IT operations by coordinating specialist agents over telemetry and software dependency evidence [251]. The remaining challenge is to connect planning, code dependencies, ownership, external side effects, testing, and recovery in a versioned structure that can explain not only whether a patch succeeded but why a particular organization of work succeeded.

Table 3 : Representative applications of System Intelligence. Level denotes the primary intelligence level: I = Individual Intelligence, S = System Intelligence, and I/S = systems spanning both. Focus summarizes the principal Graph Engineering concerns: Work Organization, Agent Team, Runtime State, and System Evolution.

System	Application Domain	Level	Focus	Structural role	Evidence or artifact	Link
			Software Engineering and IT Operations			
MetaGPT [119]	End-to-end software production	S	Work/Team	SOP-derived stages assign requirements, architecture, implementation, and review to specialized roles with structured intermediate artifacts	Collaborative software generation with executable projects and role-specific artifacts	
SWE-agent [418]	Repository issue resolution	I	Work/State	Agent-computer interface constrains repository navigation, editing, commands, and test feedback within an iterative execution trajectory	Patch resolution on real repository issues with observable action and test trajectories	
OpenHands [359]	General software development	I/S	Team/State	Event-stream runtime connects code, shell, browser, observations, and delegation while preserving execution history	Reproducible platform and evaluation across software-engineering tasks	
Codex [261]	Production software engineering	I/S	Work/Team/State	Multiple coding agents execute parallel tasks in isolated worktrees with project threads, skills, review, and background automation	Deployed coding system supporting parallel long-running engineering work	
Claude Code [10]	Repository-scale coding	I/S	Work/Team/State	Agent loop combines repository tools, checkpoints, hooks, background tasks, subagents, and parallel agent teams	Long-running coding workflows and demonstrated parallel agent-team software development	
OpenCode [5]	Open coding agent	I/S	Work/Team	Primary agents delegate specialized work to configurable subagents with separate permissions, tools, and child sessions	Open implementation supporting planning, coding, review, research, and parallel delegated tasks	
Cline [50]	Parallel software development	S	Work/Team/State	Dependency-linked tasks execute in isolated worktrees; persistent teams use a shared task board, mailbox, and mission log	Parallel coding tasks, cross-session team state, automated dependency chains, and reviewable diffs	
Project ALICE [251]	Cloud incident localization	S	Work/Team/State	Specialist agents collect telemetry, construct service and code dependency evidence, and localize operational faults	Incident investigation artifacts and validation on ITBench scenarios	
			Scientific Discovery and Laboratory Automation			
SciAgents [94]	Materials discovery	S	Work/Team	Ontological knowledge structures ground specialized agents that generate, criticize, and refine scientific hypotheses	Generated hypotheses, mechanisms, design principles, and materials proposals	
The AI Scientist [226]	Automated ML research	I	Work/State	A long-running research workflow links ideation, implementation, experiments, visualization, writing, and simulated review	End-to-end generated experiments and manuscripts across multiple ML subfields	
Virtual Lab [332]	Nanobody design	S	Work/Team/State	A principal-investigator agent organizes specialist scientist agents and external computational tools through research meetings	Experimentally validated SARS-CoV-2 nanobody designs with human oversight	
Co-Scientist [99]	Scientific hypothesis generation	S	Work/Team/State	Supervisor-managed specialized agents asynchronously generate, critique, rank, and refine hypotheses under a structured research objective	Experimentally validated biomedical hypotheses and test-time scaling of hypothesis quality	
Robin [95]	Experimental biological discovery	S	Work/Team/State	Literature and data-analysis agents connect hypothesis generation, experiment proposals, laboratory results, analysis, and revised hypotheses	Lab-in-the-loop discovery and experimental validation of therapeutic candidates	
			Healthcare and Clinical Decision Support			
DeepRare [474]	Rare-disease diagnosis	S	Work/Team/State	A central host coordinates specialized phenotype, genotype, retrieval, and analysis agents while maintaining accumulated diagnostic evidence	Evaluation across heterogeneous clinical datasets with traceable evidence-supported reasoning	
AMIE [201]	Longitudinal disease management	S	Work/Team/State	Dialogue and management-reasoning agents share patient history across visits and ground evolving care plans in clinical guidelines	Multi-visit virtual OSCE evaluation against primary-care physicians	
CARE-AD [190]	Longitudinal Alzheimer risk	S	Work/Team/State	Specialized assessments aggregate multimodal evidence across clinical time points into coordinated longitudinal predictions	Retrospective prediction across multiple horizons from longitudinal EHR notes	
MAP [43]	Inpatient clinical pathways	S	Work/Team/State	Triage, diagnosis, and treatment agents encode staged responsibilities along a clinical pathway	Evaluation of multi-agent enhancement across inpatient pathway decisions	
			Enterprise Workflows and Digital Organizations			
WorkTeam [208]	Natural-language-to-workflow	S	Work/Team	Supervisor, orchestrator, and filler agents jointly transform natural-language requirements into executable workflows	Workflow generation over 3,695 real-world enterprise samples	
SOAN [396]	Nested workflow automation	S	Work/Team	Reusable structural units are incrementally encapsulated as agents in a formalized hierarchical network	Improved adaptability, fault tolerance, and execution efficiency on complex workflows	
FinRobot-ERP [417]	Financial ERP processes	S	Work/Team/State	Business-process structures coordinate specialist agents and insert operational controls around consequential transactions	Case studies in wire transfers and employee reimbursement	
Agent-Ops [324]	E-commerce SOP automation	S	Work/Team/State	SOP grooming, web execution, and document verification are assigned to cooperating components in an auditable operational chain	Production deployment across seven SOP categories with more than 1,000 account managers	
Gemini Enterprise Agentic RAG [295]	Enterprise knowledge workflows	S	Work/Team/State	Root, planning, query-rewriting, retrieval, sufficient-context, and synthesis agents iteratively coordinate multi-source information gathering	Cross-corpus enterprise retrieval with iterative sufficiency checking and public-preview deployment	
			General-Purpose Digital Agents and Personal Automation			
OpenClaw [263]	Persistent digital assistance	I/S	Team/State	A gateway manages isolated agent identities, workspaces, authentication, session stores, skills, and channel-to-agent routing	Persistent agents operating across communication channels with independent state and workspace boundaries	
Hermes Agent [257]	General task execution	I/S	Work/State	A persistent agent combines tools, delegation, memory, reusable skills, scheduled execution, and multi-platform access	Cross-session memory and agent-generated skills that retain procedures learned during previous tasks	
			Social and Economic Simulation			
AgentSociety [457]	Large-scale social simulation	S	Team/State	Large agent populations interact through a realistic shared environment and parallelized social processes	Simulations of up to 30,000 agents and intervention-based social experiments	
EconAgent [188]	Macroeconomic simulation	S	Team/State	Heterogeneous households repeatedly interact with labor and consumption markets while memory incorporates prior personal and market experience	Multi-period macroeconomic dynamics compared with rule-based and learned agents	
SRAP-Agent [137]	Public-housing allocation	S	Work/Team/State	Applicant agents, allocation rules, scarce resources, and outcomes form an explicit policy simulation and optimization process	Policy simulation and optimization for efficiency and equity	
TwinMarket [422]	Financial-market simulation	S	Team/State	Social and trading interactions connect heterogeneous agent decisions to a shared market environment and collective feedback	Emergent group behavior, bubbles, and recessions in simulated financial markets	

9.2 Scientific Discovery and Laboratory Automation

Scientific discovery is naturally structured by dependencies among hypotheses, evidence, tools, experiments, and researchers. SciAgents uses an ontological knowledge structure to ground coordinated scientific agents, while the AI

Scientist organizes ideation, implementation, experimentation, writing, and review into a long-running research process [94, 226]. The Virtual Lab makes team structure explicit through a principal-investigator agent and specialist scientist agents, and importantly connects their computational work to physical experimental validation [332].

More recent systems move toward closed scientific feedback loops. Co-Scientist assigns generation, critique, ranking, and refinement to specialized agents managed by an asynchronous supervisor [99]. Robin combines literature-search and data-analysis agents with laboratory results so that experimental evidence can directly update subsequent hypotheses [95]. These systems demonstrate increasingly sophisticated Work, Team, and Runtime State structures, but iterative hypothesis refinement should not be confused with persistent evolution of the agent organization itself. For Graph Engineering, the stronger requirement is to preserve hypotheses, negative results, data lineage, experimental interventions, and causal dependencies while allowing evidence to influence future system structure in a reproducible manner.

9.3 Healthcare and Clinical Decision Support

Healthcare exposes the need to jointly engineer specialization, longitudinal state, authority, and evidence provenance. Earlier multi-agent consultation systems such as MAC showed that several doctor agents and a supervisor can reproduce aspects of multidisciplinary diagnosis [40]. DeepRare provides a more explicit systems architecture in which a central host coordinates specialized phenotype, genotype, retrieval, and analysis agents while accumulating traceable diagnostic evidence [474]. CARE-AD and MAP similarly organize specialist reasoning across longitudinal evidence and staged clinical responsibilities [43, 190].

AMIE extends this problem from one diagnostic episode to disease management over multiple visits. Its dialogue agent maintains conversational state while a management-reasoning agent synthesizes longitudinal patient information and clinical guidelines into evolving care plans [201]. This illustrates why Runtime State in healthcare is more than conversation memory: previous symptoms, treatments, responses, investigations, and recommendations change the validity of later actions. Clinical Graph Engineering must therefore preserve provenance, uncertainty, access boundaries, and human authorization together with the task and agent structures. Graph organization can improve coordination and traceability, but it does not establish clinical correctness by itself.

9.4 Enterprise Workflows and Digital Organizations

Enterprise applications make structural constraints concrete because actions are governed by business processes, organizational roles, permissions, and consequential updates to external systems. WorkTeam assigns workflow construction to supervisor, orchestrator, and filler agents, while SOAN builds hierarchical networks by encapsulating reusable workflow structures as agents [208, 396]. FinRobot-ERP connects specialist agents to business-process models for financial operations, and Agent-Ops combines SOP refinement, web execution, and document verification in a production-oriented multi-agent pipeline [324, 417].

Enterprise knowledge access is undergoing a similar transition. Gemini Enterprise Agentic RAG decomposes multi-source retrieval into orchestration, planning, query rewriting, search, context sufficiency checking, and synthesis, and uses feedback to continue retrieval when required information remains missing [295]. Across these systems, task completion alone is insufficient. A structurally valid enterprise agent must also respect permissions, separation of duties, policy constraints, transaction boundaries, and rollback obligations. This distinction between a planned workflow and committed external state makes enterprise automation a particularly important setting for Runtime State Management and governed Graph Engineering.

9.5 General-Purpose Digital Agents and Personal Automation

A newer application class consists of persistent digital agents that are not confined to a single professional domain. OpenClaw uses a gateway to maintain separate agent identities, workspaces, authentication profiles, sessions, and channel bindings, allowing persistent agents to operate across communication surfaces while retaining explicit state boundaries [263]. Hermes Agent similarly combines system tools, delegation, scheduled execution, persistent memory, and reusable skills across sessions and platforms [257]. In these systems, the agent is no longer instantiated only for one task; it becomes a persistent computational entity with accumulated state and continuing access to external capabilities.

This persistence also exposes an important boundary of current Graph Engineering. Hermes can convert successful procedures into reusable skills and revise them after later experience, while OpenClaw can maintain several isolated agents and route interactions between users, channels, and agent identities. These mechanisms provide cross-run adaptation and persistent organization, but they do not yet amount to general structural self-evolution. The broader challenge is to determine which experiences should alter future work structures, capability assignments, or agent relations, and how such changes can be validated, versioned, and reversed.

9.6 Social and Economic Simulation

Social and economic simulation moves the graph from an internal execution mechanism to part of the phenomenon being studied. AgentSociety supports large populations of interacting agents in realistic parallel environments [457]; EconAgent models heterogeneous households whose repeated work and consumption decisions interact with macroeconomic state [188]. SRAP-Agent connects applicant decisions, allocation rules, scarce resources, and policy outcomes [137], while TwinMarket couples individual social and trading behavior to shared market feedback and emergent financial dynamics [422]. Here, Agent Team structure determines who interacts with whom, while Runtime State records how local decisions alter the environment faced by later agents.

The same structure that enables simulation also creates an epistemic risk. Emergent behavior in an agent society depends on model choice, persona construction, interaction topology, memory, prompting, and environment rules. A graph-engineered simulator can make these assumptions explicit and support topology interventions, replay, and controlled ablations, but simulated emergence should not be interpreted as evidence of real-world causality without calibration against observations and explicit uncertainty analysis.

9.7 Cross-Domain Findings

Across application domains, the maturity of Graph Engineering is uneven. Work Organization and Agent Team Engineering are already common: applications routinely decompose objectives, assign specialized roles, schedule parallel work, and define communication or dependency structures. Explicit Runtime State Management is also becoming more visible through checkpoints, longitudinal patient records, shared task boards, event streams, experimental evidence, and evolving environments. Persistent System Evolution, however, remains rare. Most systems adapt execution within a predefined organizational structure rather than permanently revising that structure from accumulated evidence.

A second trend is the practical transition from Individual to System Intelligence. Software agents provide the clearest example: systems that initially centered on one coding trajectory now expose subagents, parallel worktrees, persistent task boards, agent teams, and supervisory interfaces. Similar changes are appearing in scientific discovery, enterprise workflows, and persistent digital assistants. Nevertheless, additional agents do not by themselves produce System Intelligence. The value of the system depends on how work is decomposed, how responsibilities are assigned, how state is shared, and how failures are diagnosed and recovered.

The application evidence therefore supports a narrower distinction between being *graph-structured* and being *graph-engineered*. Contemporary systems increasingly execute through explicit work, team, and state structures, but these structures are still usually selected manually or fixed before execution. Advancing toward full Graph Engineering requires structural objectives, graph-level observability, controlled mutation, cross-structure consistency, and evidence that successful structural changes persist and transfer across tasks and time.

10 Conclusion

Large language models have rapidly evolved from standalone generators into individual agents capable of sustained interaction, tool use, and iterative execution. Yet, as tasks become more heterogeneous, interdependent, and long-horizon, the limitations of individual intelligence become increasingly clear: a single agent loop struggles to support parallel work, specialized expertise, independent verification, and persistent state. This survey argues that the next frontier is *System Intelligence*, the ability of an agent system to organize complex objectives, coordinate heterogeneous components, and maintain coherent runtime state across the task lifecycle.

To support this transition, we introduce *Graph Engineering* as a structure-centered engineering paradigm that uses graph abstractions to make system relations explicit, operational, and adaptable. We organize the literature around

three complementary graph views: work organization, agent coordination, and runtime state management. Together, these views show how graphs can be used not only to represent tasks, agents, and states, but also to schedule work, bind capabilities, trace execution, localize failures, and enable controlled evolution. Across the surveyed methods, a common lesson emerges: system-level intelligence depends less on simply adding more models or agents, and more on explicitly organizing the relations among work, actors, and state.

Despite rapid progress, Graph Engineering remains an emerging field with important open challenges, including semantic alignment, graph governance, evaluation, privacy, and safe self-improvement. We hope this survey provides a useful foundation for understanding how graph-based abstractions can support the design of more scalable, controllable, and evolvable agent systems, and for guiding future work toward graph-native infrastructure for system intelligence.

References

[1] S. Agashe, Y. Fan, A. Reyna, and X. E. Wang. Llm-coordination: Evaluating and analyzing multi-agent coordination abilities in large language models. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 8053–8072, 2025. doi: 10.18653/v1/2025.findings-naacl.448. URL https://aclanthology.org/2025.findings-naacl.448/.

[2] L. A. Agrawal, S. Tan, D. Soylu, N. Ziems, R. Khare, K. Opsahl-Ong, A. Singhvi, H. Shandilya, M. J. Ryan, M. Jiang, C. Potts, K. Sen, A. G. Dimakis, I. Stoica, D. Klein, M. Zaharia, and O. Khattab. GEPA: Reflective prompt evolution can outperform reinforcement learning. *arXiv preprint arXiv:2507.19457*, 2025.

[3] J. Ahn and M. Kim. From prompts to contracts: Harness engineering for auditable enterprise LLM agents. *arXiv preprint arXiv:2607.08028*, 2026.

[4] P. Alonso, S. Yovine, and V. A. Braberman. Tdad: Test-driven agentic development - reducing code regressions in ai coding agents via graph-based impact analysis. *arXiv preprint arXiv:2603.17973*, 2026.

[5] Anomaly. OpenCode: Agents and subagents. OpenCode documentation, 2026. URL https://opencode.ai/docs/agents/.

[6] Anthropic. Introducing the model context protocol. Anthropic, Nov. 2024.

[7] Anthropic. Equipping agents for the real world with agent skills. Anthropic Engineering, Oct. 2025.

[8] Anthropic. Claude 3.7 sonnet and claude code. Anthropic, Feb. 2025.

[9] Anthropic. Effective harnesses for long-running agents. Anthropic Engineering, Nov. 2025.

[10] Anthropic. Claude Code: Anthropic's agentic coding system. Anthropic product documentation, 2026. URL https://www.anthropic.com/product/claude-code.

[11] Anthropic. Claude Agent SDK for Python. GitHub repository and documentation, 2026. URL https://github.com/anthropics/claude-agent-sdk-python.

[12] Apache Software Foundation. Apache Burr: Stateful application and agent framework. GitHub repository and documentation, 2026. URL https://github.com/apache/burr.

[13] A. Asai, Z. Wu, Y. Wang, A. Sil, and H. Hajishirzi. Self-RAG: Learning to retrieve, generate, and critique through self-reflection. In *International Conference on Learning Representations*, 2024.

[14] J. Bai and L. Shi. MAS-PromptBench: When does prompt optimization improve multi-agent LLM systems? *arXiv preprint arXiv:2606.23664*, 2026.

[15] T. Bai, Z. Wan, P. Zhou, X. Yu, Y. You, and I. W. Tsang. Skilldag: Self-evolving typed skill graphs for llm skill selection at scale. *arXiv preprint arXiv:2606.03056*, 2026.

[16] X. Bai, H. Lin, C. Liu, Y. Zhang, X. Jin, X. Cao, and Y. Li. SkillZip: Evaluation-free skill compression for self-evolving agents by discovering reusable structure. *arXiv preprint arXiv:2608.11079*, 2026.

[17] Y. Bai, S. Kadavath, S. Kundu, A. Askell, J. Kernion, A. Jones, A. Chen, A. Goldie, A. Mirhoseini, C. McKinnon, et al. Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*, 2022.

[18] S. Balaji, P. Mishra, A. Sachdeva, and S. Agrawal. Beyond ivr: Benchmarking customer support llm agents for business-adherence. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics*

(Industry Track), pages 193–208, 2026. doi: 10.18653/v1/2026.eacl-industry.15. URL https://aclanthology.org/2026.eacl-industry.15/.

[19] S. Barke, A. Goyal, A. Khare, A. Singh, S. Nath, and C. Bansal. AgentRx: Diagnosing AI agent failures from execution trajectories. *arXiv preprint arXiv:2602.02475*, 2026. doi: 10.48550/arXiv.2602.02475. URL https://arxiv.org/abs/2602.02475.

[20] Y. Bei, W. Zhang, S. Wang, W. Chen, S. Zhou, H. Chen, Y. Li, J. Bu, S. Pan, Y. Yu, I. King, F. Karray, and P. S. Yu. Graphs meet AI agents: Taxonomy, progress, and future opportunities. *arXiv preprint arXiv:2506.18019*, 2025. URL https://arxiv.org/abs/2506.18019.

[21] Y. Bei, T. Wei, X. Ning, Y. Zhao, Z. Liu, X. Lin, Y. Zhu, H. Hamann, J. He, and H. Tong. Mem-gallery: Benchmarking multimodal long-term conversational memory for mllm agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 40750–40784, 2026.

[22] V. Belov, A. Sosedka, A. Sakhovskiy, E. Kovtun, A. Boyarskikh, and S. Budennyy. Llm agents factory: Retrieval of domain-specific llm agents. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 4474–4479, 2026.

[23] M. Besta, N. Blach, A. Kubicek, R. Gerstenberger, M. Podstawski, L. Gianinazzi, J. Gajda, T. Lehmann, H. Niewiadomski, P. Nyczyk, and T. Hoefler. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690, 2024.

[24] A. Bonagiri, D. Borkar, G. J. Anderias, S. Rafatirad, and H. Homayoun. Causalflow: Causal attribution and counterfactual repair for llm agent failures. *arXiv preprint arXiv:2605.25338*, 2026.

[25] L. C. Borro, L. A. B. Macarini, G. Tindall, M. Montero, and A. B. Struck. Memori: A persistent memory layer for efficient, context-aware llm agents. *arXiv preprint arXiv:2603.19935*, 2026.

[26] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901, 2020.

[27] S. Cao, J. He, and F. Tan. Higmem: A hierarchical and llm-guided memory system for long-term conversational agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 33853–33862, 2026.

[28] M. Cemri, M. Z. Pan, S. Yang, L. A. Agrawal, B. Chopra, R. Tiwari, K. Keutzer, A. Parameswaran, D. Klein, K. Ramchandran, M. A. Zaharia, J. E. Gonzalez, and I. Stoica. Why do multi-agent LLM systems fail? In *Advances in Neural Information Processing Systems*, volume 38, 2025. doi: 10.52202/085713-4082.

[29] E. Y. Chang and L. Geng. Sagallm: Context management, validation, and transaction guarantees for multi-agent LLM planning. *Proceedings of the VLDB Endowment*, 18(12):4874–4886, 2025. doi: 10.14778/3750601.3750611.

[30] G. Chen, S. Dong, Y. Shu, G. Zhang, J. Sesay, B. F. Karlsson, J. Fu, and Y. Shi. Autoagents: A framework for automatic agent generation. *arXiv preprint arXiv:2309.17288*, 2023.

[31] H. Chen, X. Song, J. Jin, P. Ren, and L.-J. Zhang. Toward an organizational science of multi-agent llm systems: Decoupling who, how, and which algorithm. *arXiv preprint arXiv:2607.25446*, 2026.

[32] J. Chen, A. Prasad, S. Saha, E. Stengel-Eskin, and M. Bansal. Magicore: Multi-agent, iterative, coarse-to-fine refinement for reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 32651–32674, 2025.

[33] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021. URL https://arxiv.org/abs/2107.03374.

[34] M. Chen, J. Wang, Z. Liu, Y. Wang, and Q. Wang. From failed trajectories to reliable LLM agents: Diagnosing and repairing harness flaws. *arXiv preprint arXiv:2606.06324*, 2026.

[35] M. Chen, J. Wang, F. Mu, Y. Wang, Z. Liu, H. Feng, and Q. Wang. Seeing the whole elephant: A benchmark for failure attribution in llm-based multi-agent systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pages 19888–19905, 2026. doi: 10.18653/v1/2026.acl-long.912. URL https://aclanthology.org/2026.acl-long.912/.

[36] S. Chen, C. Zhou, Z. Yuan, Q. Zhang, Z. Cui, H. Chen, Y. Xiao, J. Cao, and X. Huang. You don't need pre-built graphs for rag: Retrieval augmented generation with adaptive reasoning structures. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 30270–30278, 2026.

[37] W. Chen, Y. Su, J. Zuo, C. Yang, C. Yuan, C.-M. Chan, H. Yu, Y. Lu, Y.-H. Hung, C. Qian, et al. Agentverse: Facilitating multi-agent collaboration and exploring emergent behaviors. In *International Conference on Learning Representations*, volume 2024, pages 20094–20136, 2024.

[38] W. Chen, Z. You, R. Li, C. Qian, C. Zhao, C. Yang, R. Xie, Z. Liu, M. Sun, et al. Internet of agents: Weaving a web of heterogeneous agents for collaborative intelligence. In *International Conference on Learning Representations*, volume 2025, pages 36374–36411, 2025.

[39] W. Chen, D. Yao, W. Li, X. Meng, C. Gong, and J. Bi. Gtool: Graph enhanced tool planning with large language model. In *International Conference on Learning Representations*, volume 2026, pages 69247–69269, 2026.

[40] X. Chen, H. Yi, M. You, W. Liu, L. Wang, H. Li, X. Zhang, Y. Guo, L. Fan, G. Chen, Q. Lao, W. Fu, K. Li, and J. Li. Enhancing diagnostic capability with multi-agents conversational large language models. *npj Digital Medicine*, 8:159, 2025. doi: 10.1038/s41746-025-01550-0. URL https://doi.org/10.1038/s41746-025-01550-0.

[41] Y. Chen, H. Lai, Y. Feng, C. Han, Q. Zhang, B. Lu, M. Li, X. Wang, Z. Wang, S. Xu, Z. Li, Z. Jin, H. Wu, C. Li, and Q. Chen. Beyond semantic organization: Memory as execution state management for long-horizon agents. *arXiv preprint arXiv:2606.06090*, 2026.

[42] Z. Chen, H. Liu, D. Xu, D. Dong, J. Li, B. Pu, and J. Zhai. Cordon: Semantic transactions for tool-using llm agents. *arXiv preprint arXiv:2606.17573*, 2026.

[43] Z. Chen, Z. Peng, X. Liang, C. Wang, P. Liang, L. Zeng, M. Ju, and Y. Yuan. MAP: Evaluation and multi-agent enhancement of large language models for inpatient pathways. *npj Health Systems*, 3:37, 2026. doi: 10.1038/s44401-026-00085-0. URL https://doi.org/10.1038/s44401-026-00085-0.

[44] Z. Chen, Q. Zhang, Z. Xiang, Z. Wei, L. Gao, X. Huang, Z. Zhang, and J. Su. Legalgraphrag: Multi-agent graph retrieval-augmented generation for reliable legal reasoning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 37455–37484, 2026.

[45] M. Cheng, J. Ouyang, S. Yu, R. Yan, Y. Luo, Z. Liu, D. Wang, Q. Liu, and E. Chen. Agent-r1: Training powerful llm agents with end-to-end reinforcement learning. *arXiv preprint arXiv:2511.14460*, 2025.

[46] P. Chhikara, D. Khant, S. Aryan, T. Singh, and D. Yadav. Mem0: Building production-ready AI agents with scalable long-term memory. *arXiv preprint arXiv:2504.19413*, 2025.

[47] N. Chirkova, T. Formal, V. Nikoulina, and S. Clinchant. Provence: Efficient and robust context pruning for retrieval-augmented generation. In *International Conference on Learning Representations*, 2025.

[48] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. PaLM: Scaling language modeling with pathways. *Journal of Machine Learning Research*, 24(240):1–113, 2023.

[49] H. W. Chung, L. Hou, S. Longpre, B. Zoph, Y. Tay, W. Fedus, Y. Li, X. Wang, M. Dehghani, S. Brahma, A. Webson, S. S. Gu, Z. Dai, M. Suzgun, X. Chen, A. Chowdhery, et al. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53, 2024.

[50] Cline. Cline: Multi-agent teams. Cline documentation, 2026. URL https://docs.cline.bot/sdk/guides/multi-agent-teams.

[51] CrewAI, Inc. CrewAI: Multi-agent automation framework. GitHub repository and documentation, 2026. URL https://github.com/crewAIInc/crewAI.

[52] D. Dai, C. Deng, C. Zhao, R. Xu, H. Gao, D. Chen, J. Li, W. Zeng, X. Yu, Y. Wu, Z. Xie, Y. K. Li, P. Huang, F. Luo, C. Ruan, Z. Sui, and W. Liang. DeepSeekMoE: Towards ultimate expert specialization in mixture-of-experts language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1280–1297. Association for Computational Linguistics, 2024.

[53] Y. Dang, C. Qian, X. Luo, J. Fan, Z. Xie, R. Shi, W. Chen, C. Yang, X. Che, Y. Tian, et al. Multi-agent collaboration via evolving orchestration. *Advances in neural information processing systems*, 38:165025–165059, 2026.

[54] S. O. de Macedo. What makes a harness a harness: Necessary and sufficient conditions for an agent harness. *arXiv preprint arXiv:2606.10106*, 2026.

[55] E. Debenedetti, J. Zhang, M. Balunović, L. Beurer-Kellner, M. Fischer, and F. Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. *arXiv preprint arXiv:2406.13352*, 2024.

[56] E. Debenedetti, I. Shumailov, T. Fan, J. Hayes, N. Carlini, D. Fabian, C. Kern, C. Shi, A. Terzis, and F. Tramèr. Defeating prompt injections by design. *arXiv preprint arXiv:2503.18813*, 2025.

[57] DeepSeek-AI. DeepSeek-V3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

[58] deepset. Haystack: Open-source ai orchestration framework. GitHub repository and documentation, 2026. URL https://github.com/deepset-ai/haystack.

[59] H. Ding, P. Liu, J. Wang, Z. Ji, M. Cao, R. Zhang, L. Ai, E. Yang, T. Shi, and L. Yu. DynaWeb: Model-based reinforcement learning of web agents. *arXiv preprint arXiv:2601.22149*, 2026.

[60] V. Dochkina. Drop the hierarchy and roles: How self-organizing llm agents outperform designed structures. *arXiv preprint arXiv:2603.28990*, 2026.

[61] G. Dong, H. Yuan, K. Lu, C. Li, M. Xue, D. Liu, W. Wang, Z. Yuan, C. Zhou, and J. Zhou. How abilities in large language models are affected by supervised fine-tuning data composition. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 177–198, 2024.

[62] G. Dong, L. Bao, Z. Wang, K. Zhao, X. Li, J. Jin, J. Yang, H. Mao, F. Zhang, K. Gai, et al. Agentic entropy-balanced policy optimization. *arXiv preprint arXiv:2510.14545*, 2025.

[63] G. Dong, J. Lu, J. Huang, W. Zhong, L. Liu, S. Huang, Z. Li, Y. Zhao, X. Song, X. Li, et al. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026.

[64] G. Dong, X. Song, Y. Hu, J. Jin, C. Zhang, Y. Chen, X. Li, H. Yuan, X. Yang, T. Wen, et al. Towards long-horizon agents: A survey. 2026.

[65] J.-K. Dong, I.-W. Huang, C.-T. Wu, and Y.-t. Tsai. Etom: A five-level benchmark for evaluating tool orchestration within the mcp ecosystem. In *Findings of the Association for Computational Linguistics: EACL 2026*, pages 1453–1488, 2026. doi: 10.18653/v1/2026.findings-eacl.75. URL https://aclanthology.org/2026.findings-eacl.75/.

[66] Y. Dong, X. Zhu, Z. Pan, L. Zhu, and Y. Yang. Villageragent: A graph-based multi-agent framework for coordinating complex task dependencies in minecraft. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024. doi: 10.18653/v1/2024.findings-acl.964. URL https://aclanthology.org/2024.findings-acl.964/.

[67] Y. Dong, J. He, Y. Hou, D. Du, Z. Xu, S. Yu, Y. Xia, and H. Chen. DeltaBox: Scaling stateful AI agents with millisecond-level sandbox checkpoint/rollback. *arXiv preprint arXiv:2605.22781*, 2026. doi: 10.48550/arXiv.2605.22781. URL https://arxiv.org/abs/2605.22781.

[68] Y. Du, Y. Wang, H. Xu, J. Xu, S. Tan, B. Zhao, B. Yang, Z. Xu, M. Kong, H. Wei, J. Liu, and Q. Zhu. Living-harness is an interactive-agent evolver. *arXiv preprint arXiv:2607.26598*, 2026.

[69] W. Duan, J. Lu, and J. Xuan. Bayesian ego-graph inference for networked multi-agent reinforcement learning. *Advances in Neural Information Processing Systems*, 38:73072–73105, 2026.

[70] D. Edge, H. Trinh, N. Cheng, J. Bradley, A. Chao, A. Mody, S. Truitt, and J. Larson. From local to global: A graph RAG approach to query-focused summarization. *arXiv preprint arXiv:2404.16130*, 2024.

[71] A. Ekelhart, K. Kurniawan, F. J. Ekaputra, and E. Kiesling. Agento: An ontology for modeling agentic ai systems. In *European Semantic Web Conference*, pages 298–320. Springer, 2026.

[72] C. Emde, A. Rubinstein, A. Goel, A. Heakl, S. Yun, S. J. Oh, and M. Gubri. MASEval: Extending multi-agent evaluation from models to systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 345–356. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-demo.34.

[73] EvoMap. Evomap: From "prompt engineering" to "epigenetic engineering". URL https://evomap.ai/blog/epigenetic-engineering.

[74] R. Fang, Y. Liang, X. Wang, J. Wu, S. Qiao, P. Xie, F. Huang, H. Chen, and N. Zhang. Memp: Exploring agent procedural memory. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 17490–17502, 2026.

[75] W. Fedus, B. Zoph, and N. Shazeer. Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39, 2022.

[76] J. Feng, S. Huang, X. Qu, G. Zhang, Y. Qin, B. Zhong, C. Jiang, J. Chi, and W. Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025.

[77] J. Feng, Y. Yu, A. Dong, X. Hu, S. Niu, P. Li, Y. Dang, A. Abdelhameed, J. Bian, X. Jiang, et al. Ontocodex: a multi-agent biomedical ontology enrichment framework. *npj Health Systems*, 3(1):73, 2026.

[78] L. Feng, Z. Xue, T. Liu, and B. An. ToRL: Scaling tool-integrated reinforcement learning. *arXiv preprint arXiv:2503.23383*, 2025.

[79] S. Feng, Z. Wang, P. Goyal, Y. Wang, W. Shi, H. Xia, H. Palangi, L. Zettlemoyer, Y. Tsvetkov, C.-Y. Lee, et al. Heterogeneous swarms: Jointly optimizing model roles and weights for multi-llm systems. *Advances in Neural Information Processing Systems*, 38:114319–114351, 2026.

[80] T. Feng, H. Zhang, Z. Lei, P. Han, and J. You. Graphplanner: Graph memory-augmented agentic routing for multi-agent llms. *arXiv preprint arXiv:2604.23626*, 2026.

[81] X. Feng, X. Song, L. Li, G. Liu, and J. Shao. Searl: Joint optimization of policy and tool graph memory for self-evolving agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24518–24535, 2026.

[82] Y. Feng, J. Sun, Z. Yang, J. Ai, C. Li, Z. Li, F. Zhang, K. He, R. Ma, J. Lin, J. Sun, Y. Xiao, S. Zhou, W. Wu, Y. Liu, P. Liu, S. Zhang, and K. Zhang. LongCLI-Bench: A preliminary benchmark and study for long-horizon agentic programming in command-line interfaces. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 29952–29963. Association for Computational Linguistics, 2026.

[83] C. Fernando, D. Banarse, H. Michalewski, S. Osindero, and T. Rocktäschel. Promptbreeder: Self-referential self-improvement via prompt evolution. In *International Conference on Learning Representations*, 2024.

[84] A. Fourney, G. Bansal, H. Mozannar, C. Tan, E. Salinas, F. Niedtner, G. Proebsting, G. Bassman, J. Gerrits, J. Alber, et al. Magentic-one: A generalist multi-agent system for solving complex tasks. *arXiv preprint arXiv:2411.04468*, 2024.

[85] D. Fu, J. Mei, R. Wu, X. Yang, J. Xu, D. Wang, P. Cai, Y. Liu, L. Wen, and B. Shi. The agent's first day: Benchmarking learning, exploration, and scheduling in the workplace scenarios. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 30094–30109, 2026.

[86] D. Fu, J. Mei, R. Wu, X. Yang, J. Xu, D. Wang, P. Cai, Y. Liu, L. Wen, and B. Shi. The agent's first day: Benchmarking learning, exploration, and scheduling in the workplace scenarios. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 30094–30109. Association for Computational Linguistics, 2026.

[87] K. Fu, L. Lyu, S. Li, S. Huang, S. Xu, J. Zheng, X. Liu, S. Liu, G. Barbone, Y. Liu, et al. Agentic laboratories of the future: Towards world models for scientific discovery. 2026.

[88] Y. Fu, R. Fang, J. Shao, H. Zheng, Z. Zhu, B. Luo, and T. Lin. Do more agents help? controlled and protocol-aligned evaluation of LLM agent workflows. *arXiv preprint arXiv:2606.05670*, 2026. URL https://arxiv.org/abs/2606.05670.

[89] M. Galster, S. Mohsenimofidi, J. L. Lulla, M. A. Abubakar, C. Treude, and S. Baltes. Configuring agentic AI coding tools: An exploratory study. *arXiv preprint arXiv:2602.14690*, 2026.

[90] H.-a. Gao, J. Geng, W. Hua, M. Hu, X. Juan, H. Liu, S. Liu, J. Qiu, X. Qi, Y. Wu, H. Wang, H. Xiao, Y. Zhou, S. Zhang, J. Zhang, J. Xiang, Y. Fang, Q. Zhao, D. Liu, Q. Ren, C. Qian, Z. Wang, M. Hu, H. Wang, Q. Wu, H. Ji, and M. Wang. A survey of self-evolving agents: What, when, how, and where to evolve on the path to artificial super intelligence. *Transactions on Machine Learning Research*, 2026. URL https://arxiv.org/abs/2507.21046.

[91] J. Gao, X. Zou, Y. Ai, D. Li, Y. Niu, B. Qi, and J. Liu. Graph counselor: Adaptive graph exploration via multi-agent synergy to enhance llm reasoning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24650–24668, 2025.

[92] L. Gao, X. Ma, J. Lin, and J. Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1762–1777. Association for Computational Linguistics, 2023.

[93] L. Geng and E. Y. Chang. Alas: Transactional and dynamic multi-agent llm planning. *arXiv preprint arXiv:2511.03094*, 2025.

[94] A. Ghafarollahi and M. J. Buehler. SciAgents: Automating scientific discovery through bioinspired multi-agent intelligent graph reasoning. *Advanced Materials*, 37(22):2413523, 2025. doi: 10.1002/adma.202413523. URL https://doi.org/10.1002/adma.202413523.

[95] A. E. Ghareeb, B. Chang, L. Mitchener, A. Yiu, C. J. Szostkiewicz, D. Shved, G. J. Gyimesi, J. M. Laurent, S. M. Wright, M. T. Razzak, A. D. White, S. C. Finnemann, M. M. Hinks, and S. G. Rodriques. A multi-agent system for automating scientific discovery. *Nature*, 655:497–505, 2026. doi: 10.1038/s41586-026-10652-y. URL https://doi.org/10.1038/s41586-026-10652-y.

[96] GitHub. Github copilot: Meet the new coding agent. GitHub Blog, May 2025.

[97] Google. Gemini cli: Your open-source ai agent. Google, June 2025.

[98] Google. Agent Development Kit: An open-source framework for ai agents. GitHub repository and documentation, 2026. URL https://github.com/google/adk-python.

[99] J. Gottweis, W.-H. Weng, A. Daryin, T. Tu, P. Sirkovic, A. Myaskovsky, G. Glowaty, F. Weissenberger, A. Orlandi, et al. Accelerating scientific discovery with Co-Scientist. *Nature*, 655:487–496, 2026. doi: 10.1038/s41586-026-10644-y. URL https://doi.org/10.1038/s41586-026-10644-y.

[100] Z. Gou et al. CRITIC: Large language models can self-correct with tool-interactive critiquing. *arXiv preprint arXiv:2305.11738*, 2023. doi: 10.48550/arXiv.2305.11738. URL https://arxiv.org/abs/2305.11738.

[101] A. Grattafiori, A. Dubey, A. Jauhri, A. Pandey, A. Kadian, A. Al-Dahle, A. Letman, A. Mathur, A. Schelten, A. Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[102] F. Grötschla, L. Müller, J. Tönshoff, M. Galkin, and B. Perozzi. Agentsnet: Coordination and collaborative reasoning in multi-agent llms. *arXiv preprint arXiv:2507.08616*, 2025. URL https://arxiv.org/abs/2507.08616.

[103] T. R. Gruber. A translation approach to portable ontology specifications. *Knowledge acquisition*, 5(2):199–220, 1993.

[104] J.-C. Gu, J. Zhang, D. Wu, Y. Li, K.-W. Chang, and N. Peng. BRIEF-Pro: Universal context compression with short-to-long synthesis for fast and accurate multi-hop reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 14221–14241. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.696.

[105] D. Guo, D. Yang, H. Zhang, J. Song, R. Zhang, R. Xu, Q. Zhu, S. Ma, P. Wang, X. Bi, et al. DeepSeek-R1 incentivizes reasoning in LLMs through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

[106] D. Guo, J. Wu, and S. M. Yiu. When to retrieve during reasoning: Adaptive retrieval for large reasoning models. *arXiv preprint arXiv:2604.26649*, 2026.

[107] J. Guo, Z. Hao, C. Wang, C. Fan, T. Luo, H. Li, Y. Gao, H. Mei, J. Peng, R. Xu, M. Dong, H. Wu, M. Zheng, K. Han, S. Wang, C. Xu, and Y. Wang. From question answering to task completion: A survey on agent system and harness design, 2026.

[108] Q. Guo, R. Wang, J. Guo, B. Li, K. Song, X. Tan, G. Liu, J. Bian, and Y. Yang. Connecting large language models with evolutionary algorithms yields powerful prompt optimizers. In *International Conference on Learning Representations*, 2024.

[109] S. Guo, Y. Wang, Z. Su, Y. Pan, Q. Hu, and T. H. Luan. Agent discovery in internet of agents: Challenges and solutions. *IEEE Network*, 2026.

[110] X. Guo, X. Wang, Y. Chen, S. Li, C. Han, M. Li, and H. Ji. Syncmind: Measuring agent out-of-sync recovery in collaborative software engineering. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 20993–21066, 2025. URL https://proceedings.mlr.press/v267/guo25l.html.

[111] J. Han, Y. Xu, Y. Liao, X. Wang, Z. Jiang, Z. Di, F. Lu, Z. Hu, and Y. Xiao. Skill-use: Can LLMs actually use skills in agentic harnesses? *arXiv preprint arXiv:2608.04828*, 2026.

[112] Z. Hao, H. Wang, J. Luo, J. Zhang, Y. Zhou, Q. Lin, C. Wang, H. Dong, and J. Chen. Recreate: Reasoning and creating domain agents driven by experience. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 31018–31046, 2026.

[113] Z. Hao, T. Wang, H. Dong, Z. Liu, H. Wang, X. Lin, Q. Lin, C. Wang, H. Dong, and J. Chen. Evolve as a team: Collaborative self-evolution for llm-based multi-agent systems. *arXiv preprint arXiv:2605.29790*, 2026.

[114] N. Hasan and P. BusiReddyGari. Dpbench: Large language models struggle with simultaneous coordination. *arXiv preprint arXiv:2602.13255*, 2026. URL https://arxiv.org/abs/2602.13255.

[115] J. He and D. Yu. Sovereign agentic loops: Decoupling AI reasoning from execution in real-world systems. *arXiv preprint arXiv:2604.22136*, 2026. doi: 10.48550/arXiv.2604.22136. URL https://arxiv.org/abs/2604.22136.

[116] Z. He, Y. Wang, C. Zhi, Y. Hu, T.-P. Chen, L. Yin, Z. Chen, T. A. Wu, S. Ouyang, Z. Wang, J. Pei, J. McAuley, Y. Choi, and A. Pentland. Memoryarena: Benchmarking agent memory in interdependent multi-session agentic tasks. *arXiv preprint arXiv:2602.16313*, 2026.

[117] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt. Measuring massive multitask language understanding. In *International Conference on Learning Representations*, 2021. URL https://arxiv.org/abs/2009.03300.

[118] J. Hoffmann, S. Borgeaud, A. Mensch, E. Buchatskaya, T. Cai, E. Rutherford, D. de Las Casas, L. A. Hendricks, J. Welbl, A. Clark, T. Hennigan, E. Noland, K. Millican, G. van den Driessche, B. Damoc, A. Guy, S. Osindero, K. Simonyan, E. Elsen, J. W. Rae, O. Vinyals, and L. Sifre. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 30016–30030, 2022.

[119] S. Hong, M. Zhuge, J. Chen, X. Zheng, Y. Cheng, J. Wang, C. Zhang, Z. Wang, S. K. S. Yau, Z. Lin, L. Zhou, C. Ran, L. Xiao, C. Wu, and J. Schmidhuber. MetaGPT: Meta programming for a multi-agent collaborative framework. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=VtmBAGCN7o.

[120] Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, and X. Huang. Next-generation database interfaces: A survey of llm-based text-to-sql. *IEEE Transactions on Knowledge and Data Engineering*, 2025.

[121] X. Hou, S. Wang, Y. Zhao, and H. Wang. When agents do not stop: Uncovering infinite agentic loops in llm agents. *arXiv preprint arXiv:2607.01641*, 2026.

[122] X. Hou, S. Wang, Y. Zhao, and H. Wang. When agents do not stop: Uncovering infinite agentic loops in LLM agents. *arXiv preprint arXiv:2607.01641*, 2026. doi: 10.48550/arXiv.2607.01641. URL https://arxiv.org/abs/2607.01641.

[123] M. Hu, T. Chen, Q. Chen, Y. Mu, W. Shao, and P. Luo. HiAgent: Hierarchical working memory management for solving long-horizon agent tasks with large language model. *arXiv preprint arXiv:2408.09559*, 2024.

[124] M. Hu, Y. Zhou, W. Fan, Y. Nie, B. Xia, T. Sun, Z. Ye, Z. Jin, Y. Li, Q. Chen, et al. Owl: Optimized workforce learning for general multi-agent assistance in real-world task automation. *arXiv preprint arXiv:2505.23885*, 2025.

[125] S. Hu, C. Lu, and J. Clune. Automated design of agentic systems. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2408.08435.

[126] W. Hu. From agent loops to structured graphs: A scheduler-theoretic framework for LLM agent execution. *arXiv preprint arXiv:2604.11378*, 2026. URL https://arxiv.org/abs/2604.11378.

[127] Y. Hu, Q. Zhou, Q. Chen, X. Li, L. Liu, D. Zhang, A. Kachroo, T. Oz, and O. Tripp. Qualityflow: An agentic workflow for program synthesis controlled by LLM quality checks. *arXiv preprint arXiv:2501.17167*, 2025. URL https://arxiv.org/abs/2501.17167.

[128] Y. Hu, Y. Wang, and J. McAuley. Evaluating memory in llm agents via incremental multi-turn interactions. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=DT7JyQC3MR.

[129] D. Huang, Y. Ding, B. Liu, Q. Liu, X. Chen, J. Bian, H. Sun, Z. Tu, D. Chu, X. Yu, and D. Sui. SkillWiki: A living knowledge infrastructure for agent skills. *arXiv preprint arXiv:2606.16523*, 2026.

[130] J. Huang, F. Cheng, J. Jiang, Z. Yu, and A. Aizawa. BenchTrace: A benchmark for testing reflection ability and controlled evolution in LLM agents. *arXiv preprint arXiv:2605.29225*, 2026. URL https://arxiv.org/abs/2605.29225.

[131] J. Huang, J. Hsia, J. Sun, F. Shi, W. Huang, and I. H. White. Proof-or-stop: Don't trust the agent, trust the evidence – loop engineering for verifiable evidence-gated lifecycle control, 2026. URL https://arxiv.org/abs/2607.14890.

[132] J. Huang, Z. Zhang, K. Shi, Y. Ye, and C. Zhang. Evolverouter: Co-evolving routing and prompt for multi-agent question answering. *arXiv preprint arXiv:2604.05149*, 2026.

[133] L. Huang, C. Yang, H. Zhou, H. Song, Z. Chen, R. Le, Y. Song, W. X. Zhao, and T. Zhang. Evo-Bench: Can language models improve agent harness? *arXiv preprint arXiv:2608.09096*, 2026.

[134] Z. Huang, H. Que, H. Zeng, G. Zhang, Z. Wang, J. Chen, H. Wang, Z. Hou, C. Pu, S. Yan, and W. Huang. Harness-IF: Evaluating instruction following across instruction surfaces in coding agents. *arXiv preprint arXiv:2608.11727*, 2026.

[135] Y. In, M. Tanjim, J. Subramanian, S. Kim, U. Bhattacharya, W. Kim, S. Park, S. Sarkhel, and C. Park. Rethinking failure attribution in multi-agent systems: A multi-perspective benchmark and evaluation. *arXiv preprint arXiv:2603.25001*, 2026. URL https://arxiv.org/abs/2603.25001.

[136] G. Izacard and E. Grave. Leveraging passage retrieval with generative models for open domain question answering. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*, pages 874–880. Association for Computational Linguistics, 2021.

[137] J. Ji, Y. Li, H. Liu, Z. Du, Z. Wei, Q. Qi, W. Shen, and Y. Lin. SRAP-Agent: Simulating and optimizing scarce resource allocation policy with LLM-based agent. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 267–293. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.findings-emnlp.15. URL https://aclanthology.org/2024.findings-emnlp.15/.

[138] S. Ji, Y. Li, and B. Hooi. Memory is reconstructed, not retrieved: Graph memory for llm agents. *arXiv preprint arXiv:2606.06036*, 2026.

[139] Q. Jia, Y. Shen, X. Song, K. Zhang, S. Wang, D. Pei, X. Zhu, and G. Zhai. One battle after another: Probing LLMs' limits on multi-turn instruction following with a benchmark evolving framework. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9574–9590. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.433.

[140] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. de las Casas, E. Bou Hanna, F. Bressand, et al. Mixtral of experts. *arXiv preprint arXiv:2401.04088*, 2024.

[141] H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13358–13376. Association for Computational Linguistics, 2023.

[142] S. Jiang, L. Ma, Z. Hong, K. Wang, Z. Lu, T. Wang, S. Chen, J. Zhang, T. Pan, W. Li, J. Liang, and Y. Xiao. SEA-Eval: A benchmark for evaluating self-evolving agents beyond episodic assessment. *arXiv preprint arXiv:2604.08988*, 2026.

[143] Z. Jiang, F. Huang, H. Xing, X. Wu, Y. Gao, R. Cao, M. Wang, S. Liu, and Y. Li. Demystifying agent skills: Why they work—until they don't. *arXiv preprint arXiv:2608.14036*, 2026.

[144] C. E. Jimenez, J. Yang, A. Wettig, S. Yao, K. Pei, O. Press, and K. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2310.06770.

[145] B. Jin, H. Zeng, Z. Yue, D. Wang, H. Zamani, and J. Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

[146] Y. Jin, K. Sharma, V. Rakesh, Y. Dou, M. Pan, M. Das, and S. Kumar. SARA: Selective and adaptive retrieval-augmented generation with context compression. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14508–14528. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.661.

[147] A. Kadu and A. Krishnan. Reflexgrad: Within-episode failure recovery in llm agents via progress-gated dual-process routing. *arXiv preprint arXiv:2511.14584*, 2025.

[148] J. Kang, M. Ji, Z. Zhao, and T. Bai. Memory OS of AI agent. *arXiv preprint arXiv:2506.06326*, 2025.

[149] M. Kang, W.-N. Chen, D. Han, H. A. Inan, L. Wutschitz, Y. Chen, R. Sim, and S. Rajmohan. ACON: Optimizing context compression for long-horizon LLM agents. In *International Conference on Machine Learning*, 2026. arXiv:2510.00615.

[150] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[151] E. Karpas, O. Abend, Y. Belinkov, B. Lenz, O. Lieber, N. Ratner, Y. Shoham, H. Bata, Y. Levine, K. Leyton-Brown, D. Muhlgay, N. Rozen, E. Schwartz, G. Shachaf, S. Shalev-Shwartz, A. Shashua, and M. Tenenholtz. MRKL systems: A modular, neuro-symbolic architecture that combines large language models, external knowledge sources and discrete reasoning. *arXiv preprint arXiv:2205.00445*, 2022.

[152] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 6769–6781. Association for Computational Linguistics, 2020.

[153] I. Kavathekar, H. Jain, A. Rathod, P. Kumaraguru, and T. Ganu. Tamas: Benchmarking adversarial risks in multi-agent llm systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 31238–31268, 2026. doi: 10.18653/v1/2026.acl-long.1442. URL https://aclanthology.org/2026.acl-long.1442/.

[154] Z. Ke, Y. Ming, A. Xu, R. Chin, X.-P. Nguyen, P. Jwalapuram, J. Wang, S. Yavuz, C. Xiong, and S. Joty. Mas-orchestra: Understanding and improving multi-agent reasoning through holistic orchestration and controlled benchmarks. *arXiv preprint arXiv:2601.14652*, 2026.

[155] S. Khan. Verified detection and prevention of concurrency anomalies in multi-agent large language model systems. *arXiv preprint arXiv:2606.17182*, 2026.

[156] S. Kim, S. Moon, R. Tabrizi, N. Lee, M. W. Mahoney, K. Keutzer, and A. Gholami. An LLM compiler for parallel function calling. In *International Conference on Machine Learning*, 2024. URL https://arxiv.org/abs/2312.04511.

[157] S. Kim, I. Bang, S. Jang, C. Kim, S. Bae, J. Choi, R. Xuan, and T. Kim. OMHBench: Benchmarking balanced and grounded omni-modal multi-hop reasoning. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 18311–18334. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.911.

[158] Y. Kim, A. Abdelaziz, T. C. Ferreira, M. Al-Badrashiny, and H. Sawaf. Bel esprit: Multi-agent framework for building ai model pipelines. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 329–339, 2025.

[159] Y. Kim, K. Gu, C. Park, C. Park, S. Schmidgall, A. A. Heydari, Y. Yan, Z. Zhang, Y. Zhuang, Y. Liu, et al. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

[160] Kimi Team. Kimi K2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

[161] A. Kotliarskyi, V. Zhu, and Z. Brock. An open-source spec for codex orchestration: Symphony. OpenAI Engineering, Apr. 2026.

[162] B. Krause, L. Chen, and E. Kahembwe. Autograms: Autonomous graphical agent modeling software. *arXiv preprint arXiv:2407.10049*, 2024.

[163] W. Kwon, Z. Li, S. Zhuang, Y. Sheng, L. Zheng, C. H. Yu, J. E. Gonzalez, H. Zhang, and I. Stoica. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, 2023. URL https://arxiv.org/abs/2309.06180.

[164] N. Lambert, J. Morrison, V. Pyatkin, S. Huang, H. Ivison, F. Brahman, L. J. V. Miranda, A. Liu, N. Dziri, S. Lyu, Y. Gu, S. Malik, V. Graf, J. D. Hwang, J. Yang, R. Le Bras, O. Tafjord, C. Wilhelm, L. Soldaini, N. A. Smith, Y. Wang, P. Dasigi, and H. Hajishirzi. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

[165] LangChain, Inc. LangChain: Agent and application framework. GitHub repository and documentation, 2026. URL https://github.com/langchain-ai/langchain.

[166] LangChain, Inc. LangGraph: Low-level orchestration for stateful agents. GitHub repository and documentation, 2026. URL https://github.com/langchain-ai/langgraph.

[167] Langflow. Langflow: Visual framework for ai agents and workflows. GitHub repository and documentation, 2026. URL https://github.com/langflow-ai/langflow.

[168] LangGenius, Inc. Dify: Agentic workflow and llm application platform. GitHub repository and documentation, 2026. URL https://github.com/langgenius/dify.

[169] H. Lee, S. Phatale, H. Mansoor, T. Mesnard, J. Ferret, K. Lu, C. Bishop, E. Hall, V. Carbune, A. Rastogi, and S. Prakash. RLAIF vs. RLHF: Scaling reinforcement learning from human feedback with AI feedback. *arXiv preprint arXiv:2309.00267*, 2023.

[170] J. Lee. Llm agents: A survey, 2026. Preprints.org preprint, posted August 5, 2026.

[171] K. Lee, D. Ippolito, A. Nystrom, C. Zhang, D. Eck, C. Callison-Burch, and N. Carlini. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445. Association for Computational Linguistics, 2022.

[172] Y. Lee, R. Nair, Q. Zhang, K. Lee, O. Khattab, and C. Finn. Meta-harness: End-to-end optimization of model harnesses. *arXiv preprint arXiv:2603.28052*, 2026.

[173] Y. Lee, H. Yen, X. Ye, and D. Chen. Agentic aggregation for parallel scaling of long-horizon agentic tasks. *arXiv preprint arXiv:2604.11753*, 2026.

[174] H. Y. Leong, Y. Li, Y. Wu, W. Ouyang, W. Zhu, J. Gao, and W. Han. Amas: Adaptively determining communication topology for llm-based multi-agent system. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 2061–2070, 2025.

[175] Letta. Letta Agent SDK: Stateful agents with persistent memory. GitHub repository and documentation, 2026. URL https://github.com/letta-ai/letta-agent-sdk.

[176] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, S. Riedel, and D. Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474, 2020.

[177] A. Li, Y. Xie, S. Li, F. Tsung, B. Ding, and Y. Li. Agent-oriented planning in multi-agent systems. In *International Conference on Learning Representations*, volume 2025, pages 19495–19517, 2025.

[178] B. Li, Z. Zhao, D.-H. Lee, and G. Wang. Adaptive graph pruning for multi-agent communication. *arXiv preprint arXiv:2506.02951*, 2025.

[179] D. Li, Z. Li, H. Du, X. Wu, S. Gui, Y. Kuang, and L. Sun. Graph of skills: Dependency-aware structural retrieval for massive agent skills. *arXiv preprint arXiv:2604.05333*, 2026.

[180] D. Li, Z. Liu, J. Wang, J. Huang, F. Li, B. Jia, B. Hu, and M. Zhang. Lycheememory v2: Efficient long-term memory for llm agents via semantic segment-level consolidation. *arXiv preprint arXiv:2608.12990*, 2026.

[181] F. Li, J. Wu, T. Fu, N. Jaques, W. Zhou, and M.-Y. Kan. Flowsteer: Prompt-only workflow steering exposes planning-time vulnerabilities in multi-agent LLM systems. *arXiv preprint arXiv:2605.11514*, 2026. URL https://arxiv.org/abs/2605.11514.

[182] G. Li, H. A. A. K. Hammoud, H. Itani, D. Khizbullin, and B. Ghanem. CAMEL: Communicative agents for "mind" exploration of large language model society. *arXiv preprint arXiv:2303.17760*, 2023. URL https://arxiv.org/abs/2303.17760.

[183] J. Li, A. Fang, G. Smyrnis, M. Ivgi, M. Jordan, S. Gadre, H. Bansal, E. Guha, S. Keh, K. Arora, S. Garg, R. Xin, N. Muennighoff, et al. DataComp-LM: In search of the next generation of training sets for language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 14200–14282, 2024.

[184] J. Li, D. Garijo, and M. Poveda-Villalón. Large language models for ontology engineering: a systematic literature review. *Semantic Web*, 17(4):22104968261465514, 2026.

[185] K. Li, J. Shi, Y. Xiao, M. Jiang, J. Sun, Y. Wu, D. Fu, S. Xia, X. Cai, T. Xu, W. Si, W. Li, D. Wang, and P. Liu. AgencyBench: Benchmarking the frontiers of autonomous agents in 1m-token real-world contexts. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pages 7422–7440. Association for Computational Linguistics, 2026.

[186] K. Li, X. Yu, Z. Ni, Y. Zeng, Y. Xu, Z. Zhang, X. Li, J. Sang, X. Duan, X. Wang, et al. Timem: Temporal-hierarchical memory consolidation for long-horizon conversational agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 21700–21720, 2026.

[187] M. Li, Y. Zhao, B. Yu, F. Song, H. Li, H. Yu, Z. Li, F. Huang, and Y. Li. API-Bank: A comprehensive benchmark for tool-augmented LLMs. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 3102–3116. Association for Computational Linguistics, 2023.

[188] N. Li, C. Gao, M. Li, Y. Li, and Q. Liao. EconAgent: Large language model-empowered agents for simulating macroeconomic activities. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15523–15536. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.829. URL https://aclanthology.org/2024.acl-long.829/.

[189] P. Li, S. Zhang, Y. Zhang, S. He, D. van Dijk, and R. Ying. Morse: Task-oriented multi-agent system with mixture of role-subtask experts. *arXiv preprint arXiv:2608.09251*, 2026.

[190] R. Li, X. Wang, D. Berlowitz, J. Mez, H. Lin, and H. Yu. CARE-AD: A multi-agent large language model framework for alzheimer's disease prediction using longitudinal clinical notes. *npj Digital Medicine*, 8:541, 2025. doi: 10.1038/s41746-025-01940-4. URL https://doi.org/10.1038/s41746-025-01940-4.

[191] S. Li, Y. Liu, Q. Wen, C. Zhang, and S. Pan. Assemble your crew: Automatic multi-agent communication topology design via autoregressive graph generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 23142–23150, 2026.

[192] X. Li, M. Liu, and C. Yuen. LLM agent communication protocol (LACP) requires urgent standardization: A telecom-inspired protocol is necessary. In *Proceedings of the NeurIPS 2025 Workshop on AI for Next Generation Communication Networks*, 2025. doi: 10.48550/arXiv.2510.13821. URL https://arxiv.org/abs/2510.13821.

[193] X. Li, W. Jiao, J. Jin, G. Dong, J. Jin, Y. Wang, H. Wang, Y. Zhu, J.-R. Wen, Y. Lu, et al. Deepagent: A general reasoning agent with scalable toolsets. In *Proceedings of the ACM Web Conference 2026*, pages 2219–2230, 2026.

[194] X. Li, T. Lyu, Y. Yang, L. Shan, S. Yang, L. Zhang, Z. Huang, Q. Liu, and Y. Li. Escaping the context bottleneck: Active context curation for LLM agents via reinforcement learning. *arXiv preprint arXiv:2604.11462*, 2026.

[195] X. Li, Y. Wang, H. Lu, Z. Chen, M. Li, P. Song, M. Zheng, and T. Cai. Memtx: Transactional belief commit for stateful agent memory. *arXiv preprint arXiv:2607.23929*, 2026.

[196] Y. Li, S. Ping, X. Chen, X. Qi, Z. Wang, Y. Luo, and X. Zhang. Agentgit: A version control framework for reliable and scalable llm-powered multi-agent systems. *arXiv preprint arXiv:2511.00628*, 2025.

[197] Y. Li, J. Yang, Z. Zheng, Z. Hu, Y. Sui, S. Wang, Y. He, and B. Hooi. Apex: Autonomous policy exploration for self-evolving llm agents. *arXiv preprint arXiv:2605.21240*, 2026.

[198] Z. Li, S. Xu, K. Mei, W. Hua, B. Rama, O. Raheja, H. Wang, H. Zhu, and Y. Zhang. Autoflow: Automated workflow generation for large language model agents. *arXiv preprint arXiv:2407.12821*, 2024. URL https://arxiv.org/abs/2407.12821.

[199] Z. Li, Y. Mi, Z. Zhou, H. Jiang, G. Zhang, K. Wang, and J. Fang. Goal-aware identification and rectification of misinformation in multi-agent systems. In *International Conference on Learning Representations*, volume 2026, pages 24661–24687, 2026.

[200] Q. Liang, H. Wang, Z. Liang, and Y. Liu. From skill text to skill structure: The scheduling-structural-logical representation for agent skills. *arXiv preprint arXiv:2604.24026*, 2026.

[201] V. Liévin, A. Palepu, W.-H. Weng, K. Saab, D. Stutz, Y. Cheng, K. Kulkarni, S. S. Mahdavi, J. Barral, D. R. Webster, et al. Towards conversational artificial intelligence for disease management. *Nature*, 655:1292–1299, 2026. doi: 10.1038/s41586-026-10764-5. URL https://doi.org/10.1038/s41586-026-10764-5.

[202] J. Lin, S. Liu, C. Pan, L. Lin, S. Dou, X. Huang, H. Yan, Z. Han, and T. Gui. Agentic harness engineering: Observability-driven automatic evolution of coding-agent harnesses. *arXiv preprint arXiv:2604.25850*, 2026.

[203] M. Lin, J. Wu, Z. Wang, Z. Shi, Y. Sang, B. He, Z. Liu, T. Wei, Z. Wu, Z. Zhang, D. Wang, X. Zhang, B. Dumoulin, C. Xie, Y. Zhou, S. Wang, and H. Lu. Harness updating is not harness benefit: Disentangling evolution capabilities in self-evolving LLM agents. *arXiv preprint arXiv:2605.30621*, 2026.

[204] A. S. Lippolis, M. J. Saeedizade, S. Schmid, S. Blattner, R. Keskisärkkä, A. Gangemi, E. Blomqvist, and A. G. Nuzzolese. Ontoextend: A framework for requirement-driven and scalable ontology extension with llms, 2026. URL https://arxiv.org/abs/2607.17963.

[205] A. Liu, J. Wang, S. Kaski, J. Wang, and M. Yang. A principle of targeted intervention for multi-agent reinforcement learning. *arXiv preprint arXiv:2510.17697*, 2025.

[206] C. Liu, C. Zhang, Y. Wu, W. Lu, N. Wu, et al. Agentpo: Enhancing multi-agent collaboration via reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pages 143134–143152, 2026.

[207] G. Liu, H. Lin, H. Zeng, H. Wang, and Q. Yao. Mas-on-the-fly: Dynamic adaptation of llm-based multi-agent systems at test time. *arXiv preprint arXiv:2602.13671*, 2026.

[208] H. Liu, R. Li, W. Xiong, Z. Zhou, and W. Peng. WorkTeam: Constructing workflows from natural language with multi-agents. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: Industry Track)*, pages 20–35. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.naacl-industry.3. URL https://aclanthology.org/2025.naacl-industry.3/.

[209] H. Liu, Y. Ming, S. Joty, and C. Zhao. Harnessing LLM agents with skill programs. *arXiv preprint arXiv:2605.17734*, 2026.

[210] J. Liu, D. Shen, Y. Zhang, B. Dolan, L. Carin, and W. Chen. What makes good in-context examples for GPT-3? In *Proceedings of Deep Learning Inside Out (DeeLIO 2022): The 3rd Workshop on Knowledge Extraction and Integration for Deep Learning Architectures*, pages 100–114. Association for Computational Linguistics, 2022.

[211] J. Liu, C. S. Xia, Y. Wang, and L. Zhang. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023. URL https://arxiv.org/abs/2305.01210.

[212] J. Liu, H. Xi, S. Zhang, Y. Zeng, T. Yue, C. Wang, J. Kang, Q. Wu, and H. Wang. Who&when pro: Can llms really attribute failures in ai agents? *arXiv preprint arXiv:2607.09996*, 2026.

[213] J. Liu, H. Xi, S. Zhang, Y. Zeng, T. Yue, C. Wang, J. Kang, Q. Wu, and H. Wang. Who&when pro: Can llms really attribute failures in ai agents? *arXiv preprint arXiv:2607.09996*, 2026. URL https://arxiv.org/abs/2607.09996.

[214] N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173, 2024.

[215] S. Liu, J. Yang, B. Jiang, Y. Li, J. Guo, X. Liu, and B. Dai. Context as a tool: Context management for long-horizon SWE-agents. *arXiv preprint arXiv:2512.22087*, 2025.

[216] X. Liu, H. Yu, H. Zhang, Y. Xu, X. Lei, H. Lai, Y. Gu, H. Ding, K. Men, K. Yang, S. Zhang, X. Deng, A. Zeng, Z. Du, C. Zhang, S. Shen, T. Zhang, Y. Su, H. Sun, M. Huang, Y. Dong, and J. Tang. AgentBench: Evaluating LLMs as agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2308.03688.

[217] X. Liu, R. Song, X. Wang, and X. Chen. Select, read, and write: A multi-agent framework of full-text-based related work generation. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 7009–7028, 2025.

[218] Y. Liu, G. Zhang, K. Wang, S. Li, and S. Pan. Graph-augmented large language model agents: Current progress and future prospects. *arXiv preprint arXiv:2507.21407*, 2025. URL https://arxiv.org/abs/2507.21407.

[219] Y. Liu, Y. Liu, X. Yin, B. Wang, C. Zhang, H. Yin, and Z. Han. Openclawbench: Benchmarking process-side anomalies in real-world agent execution trajectories. *arXiv preprint arXiv:2605.29253*, 2026. URL https://arxiv.org/abs/2605.29253.

[220] Z. Liu, Y. Zhang, P. Li, Y. Liu, and D. Yang. A dynamic llm-powered agent network for task-oriented agent collaboration. *arXiv preprint arXiv:2310.02170*, 2023.

[221] Z. Liu, C. Chen, W. Li, P. Qi, T. Pang, C. Du, W. S. Lee, and M. Lin. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*, 2025.

[222] Z. Liu, Z. Shi, Y. Sang, B. He, M. Lin, T. Wei, D. Wang, B. Dumoulin, W. Jin, and H. Lu. Adaptive auto-harness: Sustained self-improvement for agentic system deployment on open-ended task streams. *arXiv preprint arXiv:2606.01770*, 2026.

[223] LlamaIndex. LlamaIndex Workflows: Event-driven agent workflows. GitHub repository and documentation, 2026. URL https://github.com/run-llama/workflows-py.

[224] S. Longpre, L. Hou, T. Vu, A. Webson, H. W. Chung, Y. Tay, D. Zhou, Q. V. Le, B. Zoph, J. Wei, and A. Roberts. The flan collection: Designing data and methods for effective instruction tuning. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pages 22631–22648. PMLR, 2023.

[225] R. Lopopolo. Harness engineering: Leveraging codex in an agent-first world. OpenAI Engineering, Feb. 2026.

[226] C. Lu, C. Lu, R. T. Lange, J. Foerster, J. Clune, and D. Ha. The ai scientist: Towards fully automated open-ended scientific discovery. *arXiv preprint arXiv:2408.06292*, 2024. doi: 10.48550/arXiv.2408.06292. URL https://arxiv.org/abs/2408.06292.

[227] J. Lu, T. Holleis, Y. Zhang, B. Aumayer, F. Nan, F. Bai, S. Ma, S. Ma, M. Li, G. Yin, Z. Wang, and R. Pang. ToolSandbox: A stateful, conversational, interactive evaluation benchmark for LLM tool use capabilities. *arXiv preprint arXiv:2408.04682*, 2024.

[228] Y. Lu, Y. Hu, X. Zhao, and J. Cao. Dytopo: Dynamic topology routing for multi-agent reasoning via semantic matching. *arXiv preprint arXiv:2602.06039*, 2026.

[229] X. Luo, Y. Zhang, Z. He, Z. Wang, S. Zhao, D. Li, L. K. Qiu, and Y. Yang. Agent lightning: Train ANY AI agents with reinforcement learning. *arXiv preprint arXiv:2508.03680*, 2025.

[230] Y. Luo, R. Gao, L. Teng, X. Wen, J. Jiang, Q. Zhang, Y. Sun, S. Zhang, J. Feng, T. Liu, W. Zhang, and D. Pei. Graph of states: Solving abductive tasks with large language models. In *International Conference on Machine Learning*, 2026. URL https://arxiv.org/abs/2603.21250.

[231] C. Ma, J. Zhang, Z. Zhu, C. Yang, Y. Yang, Y. Jin, Z. Lan, L. Kong, and J. He. AgentBoard: An analytical evaluation board of multi-turn LLM agents. *arXiv preprint arXiv:2401.13178*, 2024. doi: 10.48550/arXiv.2401.13178. URL https://arxiv.org/abs/2401.13178.

[232] Y. Ma, Z. Wang, Y. Li, Z. Li, X. Guo, W. Sun, C. Zhang, and Y. Ye. Proplay: Procedural world models for self-evolving llm agents. *arXiv preprint arXiv:2606.12780*, 2026.

[233] Z. Ma, H. Huang, S. Zou, Y. Wang, S. Yang, Y. Hu, F. Wei, and X. Chu. Longhorizon-harness: Advancing long-horizon agents for real-world tasks. *arXiv preprint arXiv:2608.01964*, 2026.

[234] S. Macedo. Stop hand-holding your coding agent: Engineering the loops that replace step-by-step prompting, 2026. URL https://arxiv.org/abs/2607.00038.

[235] S. Macedo. Stop hand-holding your coding agent: Engineering the loops that replace step-by-step prompting. *arXiv preprint arXiv:2607.00038*, 2026. doi: 10.48550/arXiv.2607.00038. URL https://arxiv.org/abs/2607.00038.

[236] A. Madaan, N. Tandon, P. Gupta, S. Hallinan, L. Gao, S. Wiegreffe, U. Alon, N. Dziri, S. Prabhumoye, Y. Yang, S. Gupta, B. P. Majumder, K. Hermann, S. Welleck, A. Yazdanbakhsh, and P. Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594, 2023.

[237] S. Mahmud, E. Bagdasarian, and S. Zilberstein. Collab: A framework for designing scalable benchmarks for agentic llms. In *NeurIPS 2025 Workshop on Scaling Environments for Agents*, 2025. URL https://openreview.net/forum?id=372FjQy1cF.

[238] Mastra. Mastra: Typescript framework for ai agents and workflows. GitHub repository and documentation, 2026. URL https://github.com/mastra-ai/mastra.

[239] K. Mei, X. Zhu, W. Xu, W. Hua, M. Jin, Z. Li, S. Xu, R. Ye, Y. Ge, and Y. Zhang. AIOS: Llm agent operating system. In *Conference on Language Modeling*, 2025.

[240] K. Mei et al. AIOS: LLM agent operating system. *arXiv preprint arXiv:2403.16971*, 2024. doi: 10.48550/arXiv.2403.16971. URL https://arxiv.org/abs/2403.16971.

[241] T. Men, P. Cao, Z. Jin, Y. Chen, K. Liu, and J. Zhao. A troublemaker with contagious jailbreak makes chaos in honest towns. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 17561–17587, 2025.

[242] Q. Meng, Y. Wang, L. Chen, Y. Li, W. Wu, W. Jiang, Q. Wang, C. Lu, Y. Gao, Y. Wu, and Y. Hu. Agent harness for large language model agents: A survey, 2026.

[243] G. Mialon, C. Fourrier, C. Swift, T. Wolf, Y. LeCun, and T. Scialom. GAIA: A benchmark for general AI assistants. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2311.12983.

[244] Microsoft. AutoGen: A programming framework for agentic ai. GitHub repository, 2026. URL https://github.com/microsoft/autogen. Maintenance mode.

[245] Microsoft. Microsoft Agent Framework. GitHub repository and documentation, 2026. URL https://github.com/microsoft/agent-framework.

[246] S. Min, X. Lyu, A. Holtzman, M. Artetxe, M. Lewis, H. Hajishirzi, and L. Zettlemoyer. Rethinking the role of demonstrations: What makes in-context learning work? In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 11048–11064. Association for Computational Linguistics, 2022.

[247] H. Ming, F. Li, X. Wu, and W. Que. Retrieval as reasoning: Self-evolving agent-native retrieval via LLM-Wiki. *arXiv preprint arXiv:2605.25480*, 2026.

[248] Model Context Protocol Contributors. Model Context Protocol Python SDK. Official GitHub repository and documentation, 2026. URL https://github.com/modelcontextprotocol/python-sdk.

[249] B. Mohammadi, N. Potamitis, L. Klein, A. Arora, and L. Bindschaedler. Atomix: Timely, transactional tool use for reliable agentic workflows. *arXiv preprint arXiv:2602.14849*, 2026.

[250] C. Mu, Y. Zeng, Q. Zhang, K. Shao, C. Chu, H. Guo, D. Jia, Z. Wang, and S. Hu. Adaptive theory of mind for llm-based multi-agent coordination. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 29608–29616, 2026.

[251] M. Murphy. Teams of agents can take the headaches and potential costs out of finding it bugs. IBM Research, Dec. 2025. URL https://research.ibm.com/blog/project-alice-software-bugs-agents. Project ALICE: Agentic Logic for Incident and Codebug Elimination.

[252] Y. Nakajima. The log is the agent: Event-sourced reactive graphs for auditable, forkable agentic systems. *arXiv preprint arXiv:2605.21997*, 2026.

[253] A. Ni et al. LEVER: Learning to verify language-to-code generation with execution. *arXiv preprint arXiv:2302.08468*, 2023. doi: 10.48550/arXiv.2302.08468. URL https://arxiv.org/abs/2302.08468.

[254] Z. Nie, R. Shen, X. Yu, B. Yin, J. Zhang, and X. Hu. Skillgraph: Self-evolving multi-agent collaboration with multimodal graph topology. *arXiv preprint arXiv:2604.17503*, 2026.

[255] X. Ning, K. Tieu, D. Fu, et al. Code as agent harness. *arXiv preprint arXiv:2605.18747*, 2026. URL https://arxiv.org/abs/2605.18747.

[256] B. Niu, Y. Song, K. Lian, Y. Shen, Y. Yao, K. Zhang, and T. Liu. Flow: Modularized agentic workflow automation. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/ba84da6921f3040b74ee163aa7451f53-Abstract-Conference.html.

[257] Nous Research. Hermes Agent: The agent that grows with you. GitHub repository and documentation, 2026. URL https://github.com/NousResearch/hermes-agent.

[258] NVIDIA. Megatron-LM and Megatron Core: Gpu-optimized training of transformer models at scale. GitHub repository and documentation, 2026. URL https://github.com/NVIDIA/Megatron-LM.

[259] OpenAI. GPT-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[260] OpenAI. Introducing codex. OpenAI, May 2025.

[261] OpenAI. Introducing the Codex app. OpenAI, Feb. 2026. URL https://openai.com/index/introducing-the-codex-app/.

[262] OpenAI. OpenAI Agents SDK. GitHub repository and documentation, 2026. URL https://github.com/openai/openai-agents-python.

[263] OpenClaw Contributors. OpenClaw: Persistent personal agents and multi-agent routing. GitHub repository and documentation, 2026. URL https://github.com/openclaw/openclaw.

[264] K. Opsahl-Ong, M. J. Ryan, J. Purtell, D. Broman, C. Potts, M. Zaharia, and O. Khattab. Optimizing instructions and demonstrations for multi-stage language model programs. *arXiv preprint arXiv:2406.11695*, 2024.

[265] A. Orogat, A. Rostam, and E. Mansour. Understanding multi-agent llm frameworks: A unified benchmark and experimental analysis. *arXiv preprint arXiv:2602.03128*, 2026. URL https://arxiv.org/abs/2602.03128.

[266] U. Ortaç, E. Tosun, A. K. Özbek, F. B. Terzioğlu, and R. Bayraktar. Agentology: Ontology-driven operational environments for multi-agent systems. *Available at SSRN 6919461*, 2026.

[267] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. L. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, J. Schulman, J. Hilton, F. Kelton, L. Miller, M. Simens, A. Askell, P. Welinder, P. F. Christiano, J. Leike, and R. Lowe. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744, 2022.

[268] S. Ouyang, J. Yan, I. Hsu, Y. Chen, K. Jiang, Z. Wang, R. Han, L. Le, S. Daruki, X. Tang, et al. Reasoningbank: Scaling agent self-evolving with reasoning memory. In *International Conference on Learning Representations*, volume 2026, pages 94327–94354, 2026.

[269] C. Packer, V. Fang, S. G. Patil, K. Lin, S. Wooders, and J. E. Gonzalez. MemGPT: Towards LLMs as operating systems. *arXiv preprint arXiv:2310.08560*, 2023.

[270] Palantir Technologies. The ontology system, 2026. URL https://www.palantir.com/docs/foundry/architecture-center/ontology-system. Palantir Foundry Architecture Center.

[271] Q. Pan, Y. Yang, J. Li, J. Zhou, K. Chen, X. Li, Q. Chen, and L. He. Anything2Skill: Compiling external knowledge into reusable skills for agents. *arXiv preprint arXiv:2606.09316*, 2026.

[272] W. Pan, S. Liu, C.-Y. Lin, J. Zeng, X. Tang, X. Zhou, Y. Lu, and X. Jia. Evolving agents in the dark: Retrospective harness optimization via self-preference. *arXiv preprint arXiv:2606.05922*, 2026.

[273] C. Papadakis, A. Dimitriou, G. Filandrianos, M. Lymperaiou, K. Thomas, and G. Stamou. Atlas: Adaptive trading with llm agents through dynamic prompt optimization and multi-agent coordination. *arXiv preprint arXiv:2510.15949*, 2025.

[274] A. Pappu, B. El, H. Cao, C. di Nolfo, Y. Sun, M. Cao, and J. Zou. Multi-agent teams hold experts back. *arXiv preprint arXiv:2602.01011*, 2026.

[275] A. Parisi, Y. Zhao, and N. Fiedel. TALM: Tool augmented language models. *arXiv preprint arXiv:2205.12255*, 2022.

[276] J. S. Park, J. C. O'Brien, C. J. Cai, M. R. Morris, P. Liang, and M. S. Bernstein. Generative agents: Interactive simulacra of human behavior. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, 2023.

[277] S. G. Patil, T. Zhang, X. Wang, and J. E. Gonzalez. Gorilla: Large language model connected with massive APIs. In *Advances in Neural Information Processing Systems*, volume 37, 2024.

[278] G. Penedo, H. Kydlíček, L. Ben Allal, A. Lozhkov, M. Mitchell, C. Raffel, L. von Werra, and T. Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *Advances in Neural Information Processing Systems*, volume 37, pages 30811–30849, 2024.

[279] S. Perera, K. Hapuarachchi, F. Leymann, and R. Khalaf. Robust agent compensation (rac): Teaching ai agents to compensate. In *Proceedings of the ACM Conference on AI and Agentic Systems*, pages 253–262, 2026.

[280] C. Polat, M. Tuncel, M. Kurban, E. Serpedin, and H. Kurban. xchemagents: Agentic ai for explainable quantum chemistry. *arXiv preprint arXiv:2505.20574*, 2025.

[281] H. Pu, X. Song, and L. Zhao. SkillOps: Managing LLM agent skill libraries as self-maintaining software ecosystems. *arXiv preprint arXiv:2605.13716*, 2026.

[282] Pydantic Services Inc. Pydantic AI: Typed agent framework and graph runtime. GitHub repository and documentation, 2026. URL https://github.com/pydantic/pydantic-ai.

[283] J. Qi, Z. Luan, H. Zhang, S. Huang, C. Fung, Y. Tong, H. Yang, and D. Qian. Can llms really recover microservice failures? a recovery-aware evaluation of diagnosis-to-action reasoning. *arXiv preprint arXiv:2607.04623*, 2026. URL https://arxiv.org/abs/2607.04623.

[284] S. Qi, J. Ma, R. Xing, W. Guo, X. Huang, Z. Gao, J. Deng, J. Liu, L. Zhang, B. Wei, B. Yang, P. Wang, J. Sun, J. Tao, Y. Wu, H. Liu, Y. Yao, and T. Liu. Beyond individual intelligence: Surveying collaboration, failure attribution, and self-evolution in LLM-based multi-agent systems. *arXiv preprint arXiv:2605.14892*, 2026. URL https://arxiv.org/abs/2605.14892.

[285] Z. Qi, X. Liu, I. L. Iong, H. Lai, X. Sun, W. Zhao, Y. Yang, X. Yang, J. Sun, S. Yao, T. Zhang, W. Xu, J. Tang, and Y. Dong. WebRL: Training LLM web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, 2025.

[286] C. Qian, W. Liu, H. Liu, N. Chen, Y. Dang, J. Li, C. Yang, W. Chen, Y. Su, X. Cong, J. Xu, D. Li, Z. Liu, and M. Sun. ChatDev: Communicative agents for software development. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15174–15186. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-long.810. URL https://aclanthology.org/2024.acl-long.810/.

[287] C. Qian, E. C. Acikgoz, Q. He, H. Wang, X. Chen, D. Hakkani-Tür, G. Tur, and H. Ji. Toolrl: Reward is all tool learning needs. *arXiv preprint arXiv:2504.13958*, 2025.

[288] C. Qian, Z. Xie, Y. Wang, W. Liu, K. Zhu, H. Xia, Y. Dang, Z. Du, W. Chen, C. Yang, et al. Scaling large language model-based multi-agent collaboration. In *International Conference on Learning Representations*, volume 2025, pages 41488–41505, 2025.

[289] S. Qiao, R. Fang, Z. Qiu, X. Wang, N. Zhang, Y. Jiang, P. Xie, F. Huang, and H. Chen. Benchmarking agentic workflow generation. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/adbe936993aa7cf41e45054d8b72f183-Abstract-Conference.html.

[290] Y. Qin, S. Liang, Y. Ye, K. Zhu, L. Yan, Y. Lu, Y. Lin, X. Cong, X. Tang, B. Qian, S. Zhao, L. Hong, R. Tian, R. Xie, J. Zhou, M. Gerstein, D. Li, Z. Liu, and M. Sun. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *International Conference on Learning Representations*, 2024.

[291] Qwen Team. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

[292] J. W. Rae, S. Borgeaud, T. Cai, K. Millican, J. Hoffmann, F. Song, J. Aslanides, S. Henderson, R. Ring, S. Young, et al. Scaling language models: Methods, analysis & insights from training gopher. *arXiv preprint arXiv:2112.11446*, 2021.

[293] R. Rafailov, A. Sharma, E. Mitchell, C. D. Manning, S. Ermon, and C. Finn. Direct preference optimization: Your language model is secretly a reward model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741, 2023.

[294] P. Rajasekaran. Harness design for long-running application development. Anthropic Engineering, Mar. 2026.

[295] C. Rashtchian and D.-C. Juan. Unlocking dependable responses with Gemini Enterprise Agent Platform's agentic RAG. Google Research, June 2026. URL https://research.google/blog/unlocking-dependable-responses-with-gemini-enterprise-agent-platforms-agentic-rag/.

[296] P. Rasmussen, P. Paliychuk, T. Beauvais, J. Ryan, and D. Chalef. Zep: A temporal knowledge graph architecture for agent memory. *arXiv preprint arXiv:2501.13956*, 2025.

[297] J. Recker, F. Hensen, A. Keuper, and G. Schuh. Lamas4pd-a multi-agent llm approach for ontology-driven structuring of engineering knowledge in industry 4.0. *Procedia CIRP*, 142:274–279, 2026.

[298] D. Rein, B. L. Hou, A. C. Stickland, J. Petty, R. Y. Pang, J. Dirani, J. Michael, and S. R. Bowman. GPQA: A graduate-level google-proof Q&A benchmark. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2311.12022.

[299] Z. Ren, Y. Yang, Y. Chen, Z. Zhao, B. Fu, Z. Shu, B. Zhang, Y. Xu, D. Guo, and S. Yan. Gatemem: Benchmarking memory governance in multi-principal shared-memory agents. *arXiv preprint arXiv:2606.18829*, 2026. URL https://arxiv.org/abs/2606.18829.

[300] L. Reynolds and K. McDonell. Prompt programming for large language models: Beyond the few-shot paradigm. *arXiv preprint arXiv:2102.07350*, 2021.

[301] A. Rezazadeh, Z. Li, A. Lou, Y. Zhao, W. Wei, and Y. Bao. Collaborative memory: Multi-user memory sharing in LLM agents with dynamic access control. *arXiv preprint arXiv:2505.18279*, 2025.

[302] C. Riedl. Emergent coordination in multi-agent language models. In *International Conference on Learning Representations*, volume 2026, pages 120776–120799, 2026.

[303] J. Ruan, Z. Xu, Y. Peng, F. Ren, Z. Yu, X. Liang, J. Xiang, Y. Chen, B. Liu, C. Wu, et al. Aorchestra: Automating sub-agent creation for agentic orchestration. *arXiv preprint arXiv:2602.03786*, 2026.

[304] Y. Ruan, H. Dong, A. Wang, S. Pitis, Y. Zhou, J. Ba, Y. Dubois, C. J. Maddison, and T. Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. *arXiv preprint arXiv:2309.15817*, 2023.

[305] V. Sanh, A. Webson, C. Raffel, S. H. Bach, L. Sutawika, Z. Alyafeai, A. Chaffin, A. Stiegler, T. Le Scao, A. Raja, et al. Multitask prompted training enables zero-shot task generalization. In *International Conference on Learning Representations*, 2022.

[306] S. Sarin, L. Singh, B. Sarmah, and D. Mehta. Memoria: A scalable agentic memory framework for personalized conversational ai. *arXiv preprint arXiv:2512.12686*, 2025.

[307] T. Schick, J. Dwivedi-Yu, R. Dessì, R. Raileanu, M. Lomeli, E. Hambro, L. Zettlemoyer, N. Cancedda, and T. Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[308] B. Sengupta and J. Wang. HARBOR: Automated harness optimization. *arXiv preprint arXiv:2604.20938*, 2026.

[309] W. Seo, W. Choi, J. Koh, J. Lee, H. An, M. Yu, J. Park, Q. Zhou, S. Lee, and Y. Bu. Toward culturally aligned llms through ontology-guided multi-agent reasoning. *arXiv preprint arXiv:2601.21700*, 2026.

[310] R. Shahroz, Z. Tan, S. Yun, C. Fleming, and T. Chen. Agents under siege: Breaking pragmatic multi-agent llm systems with optimized prompt attacks. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 9661–9674, 2025.

[311] F. Shang and Y. Yang. Hypothesis-driven skill optimization for llm agents. *arXiv preprint arXiv:2606.22330*, 2026.

[312] Y. Shang, Y. Li, K. Zhao, L. Ma, J. Liu, F. Xu, and Y. Li. Agentsquare: Automatic llm agent search in modular design space. In *International Conference on Learning Representations*, volume 2025, pages 3841–3865, 2025.

[313] Q. Shao, L. Yuan, X. Lin, and W. Zhang. Augmenting the intelligence of large language model-based agents with graphs: A survey, 2026. Preprint.

[314] Y. Shao, V. Samuel, Y. Jiang, J. Yang, and D. Yang. Collaborative gym: A framework for enabling and evaluating human-agent collaboration. In *International Conference on Learning Representations*, volume 2026, pages 99616–99649, 2026.

[315] Z. Shao, P. Wang, Q. Zhu, R. Xu, J. Song, X. Bi, H. Zhang, M. Zhang, Y. K. Li, Y. Wu, and D. Guo. DeepSeekMath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[316] X. Shen, Y. Liu, Y. Dai, Y. Wang, R. Miao, Y. Tan, S. Pan, and X. Wang. Understanding the information propagation effects of communication topologies in llm-based multi-agent systems. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 12358–12372, 2025.

[317] Y. Shen, K. Song, X. Tan, D. Li, W. Lu, and Y. Zhuang. HuggingGPT: Solving AI tasks with ChatGPT and its friends in Hugging Face. In *Advances in Neural Information Processing Systems*, volume 36, 2023. URL https://arxiv.org/abs/2303.17580.

[318] Y. Shen, K. Song, X. Tan, W. Zhang, K. Ren, S. Yuan, W. Lu, D. Li, and Y. Zhuang. Taskbench: Benchmarking large language models for task automation. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/085185ea97db31ae6dcac7497616fd3e-Abstract-Datasets_and_Benchmarks_Track.html.

[319] Y. Shen, K. Li, W. Zhou, and S. Hu. Mem2actbench: A benchmark for evaluating long-term memory utilization in task-oriented autonomous agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pages 8173–8190, 2026. doi: 10.18653/v1/2026.acl-long.370. URL https://aclanthology.org/2026.acl-long.370/.

[320] Z. Shen, S. Cheng, Z. Guo, W. Wang, Y. Wang, and H. Huang. Anchormem: Anchored facts with associative contexts for building memory in large language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 34784–34798, 2026.

[321] G. Sheng, C. Zhang, Z. Ye, X. Wu, W. Zhang, R. Zhang, Y. Peng, H. Lin, and C. Wu. HybridFlow: A flexible and efficient RLHF framework. In *Proceedings of the Twentieth European Conference on Computer Systems*, 2025. URL https://arxiv.org/abs/2409.19256.

[322] T. Shin, Y. Razeghi, R. L. Logan IV, E. Wallace, and S. Singh. AutoPrompt: Eliciting knowledge from language models with automatically generated prompts. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*, pages 4222–4235. Association for Computational Linguistics, 2020.

[323] M. Shoeybi, M. Patwary, R. Puri, P. LeGresley, J. Casper, and B. Catanzaro. Megatron-LM: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019. URL https://arxiv.org/abs/1909.08053.

[324] A. Singh, S. Agrawal, S. Adhikari, V. S. Puranik, S. Tiwari, and D. Assudani. Agent-ops: A multi-agent orchestration framework for end-to-end SOP automation in e-commerce operations. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 436–446. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-industry.29. URL https://aclanthology.org/2026.acl-industry.29/.

[325] H. Song et al. Beyond the protocol: Unveiling attack vectors in the model context protocol (MCP) ecosystem. *IEEE Transactions on Software Engineering*, 52(8):2410–2426, 2026. doi: 10.1109/TSE.2026.3694876.

[326] K. Song, A. Jayarajan, Y. Ding, Q. Su, Z. Zhu, S. Liu, and G. Pekhimenko. Aegis: Taxonomy and optimizations for overcoming agent-environment failures in llm agents. *arXiv preprint arXiv:2508.19504*, 2025.

[327] X. Song, L. Zhang, K. Zhao, Y. Zhu, Z. Wang, G. Dong, J. Yang, H. Li, K. Gai, J.-R. Wen, et al. Webswarm: Recursive multi-agent orchestration for deep-and-wide web search. *arXiv preprint arXiv:2607.08662*, 2026.

[328] H. Sun, Y. Zhuang, L. Kong, B. Dai, and C. Zhang. AdaPlanner: Adaptive planning from feedback with language models. *arXiv preprint arXiv:2305.16653*, 2023. doi: 10.48550/arXiv.2305.16653. URL https://arxiv.org/abs/2305.16653.

[329] H. Sun, S. Zhang, L. Niu, L. Ren, H. Xu, H. Fu, F. Zhao, C. Yuan, and X. Wang. Collab-overcooked: Benchmarking and evaluating large language models as collaborative agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 4922–4951, 2025. doi: 10.18653/v1/2025.emnlp-main.249. URL https://aclanthology.org/2025.emnlp-main.249/.

[330] H. Sun, Y. Min, Z. Chen, X. Zhao, and J.-R. Wen. Challenging the boundaries of reasoning: An olympiad-level math benchmark for large language models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 17438–17457. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.792.

[331] Y. Sun, Z. Zhao, S. Wan, and C. Gong. Cortexdebate: Debating sparsely and equally for multi-agent debate. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 9503–9523, 2025.

[332] K. Swanson, W. Wu, N. L. Bulaong, J. E. Pak, and J. Zou. The virtual lab of ai agents designs new SARS-CoV-2 nanobodies. *Nature*, 646:716–723, 2025. doi: 10.1038/s41586-025-09442-9. URL https://doi.org/10.1038/s41586-025-09442-9.

[333] Y. Tang, C. Yang, S. Liu, Z. Xiang, Z. Chen, Q. Zhang, and J. Su. Saas: Self-aware reinforcement learning for over-search mitigation in agentic search. *arXiv preprint arXiv:2605.29796*, 2026.

[334] O. Team. Gpt-4 technical report, 2024. URL https://arxiv.org/abs/2303.08774.

[335] C. Tian, Y. Yao, and J. Cui. Queenbee planner: Skill-evolving communication topologies for token-efficient llm multi-agent systems. *arXiv preprint arXiv:2606.27492*, 2026.

[336] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, A. Rodriguez, A. Joulin, E. Grave, and G. Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[337] H. Trivedi, N. Balasubramanian, T. Khot, and A. Sabharwal. Interleaving retrieval with chain-of-thought reasoning for knowledge-intensive multi-step questions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 10014–10037. Association for Computational Linguistics, 2023.

[338] H. Trivedi, T. Khot, M. Hartmann, R. Manku, V. Dong, E. Li, S. Gupta, A. Sabharwal, and N. Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, 2024. URL https://arxiv.org/abs/2407.18901.

[339] V. Ursekar, A. Shanker, Y. Maurya, S. Yasser, V. S. Kalmath, V. Chatrath, and Y. Xue. HarnessOpt-Bench: Evaluating LLMs at harness optimization. *arXiv preprint arXiv:2608.06301*, 2026.

[340] N. Vats and O. Golev. The scaffold effect in coding agents: Harness choice as a hidden variable in coding-agent evaluation. *arXiv preprint arXiv:2607.22585*, 2026.

[341] G. Wan, M. Zhou, Z. Wang, X. Shang, E. H. Jiang, G. Zhang, J. Bi, Y. Ma, Z. Zhang, K. Liang, and W. Huang. DAWN: Distributed LLM multi-agent workflow synthesis. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 26099–26106, 2026. doi: 10.1609/aaai.v40i31.39812.

[342] C. Wang, Q. Wu, and AG2 Community. AG2: Open-source agentos for ai agents. GitHub repository and documentation, 2026. URL https://github.com/ag2ai/ag2.

[343] C. Wang, Z. Yu, X. Xie, W. Yao, R. Fang, S. Qiao, K. Cao, G. Zheng, X. Qi, P. Zhang, and S. Deng. SkillX: Automatically constructing skill knowledge bases for agents. *arXiv preprint arXiv:2604.04804*, 2026.

[344] G. Wang, Y. Xie, Y. Jiang, A. Mandlekar, C. Xiao, Y. Zhu, L. Fan, and A. Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

[345] H. Wang, M. Zhang, C. Yu, Y. Shang, X. Hu, G. Wang, and N. Zou. a^2e : An end-to-end agent auditing engine, 2026. URL https://arxiv.org/abs/2608.07346.

[346] J. Wang, J. Wang, B. Athiwaratkun, C. Zhang, and J. Y. Zou. Mixture-of-agents enhances large language model capabilities. In *International Conference on Learning Representations*, volume 2025, pages 33944–33963, 2025.

[347] J. Wang, Q. Yan, Y. Wang, Y. Tian, S. S. Mishra, Z. Xu, M. Gandhi, P. Xu, and L. L. Cheong. Reinforcement learning for self-improving agent with skill library. *arXiv preprint arXiv:2512.17102*, 2025.

[348] K. Wang, Y. Lin, J. Lou, Z. Zhou, B. Suvonov, and J. Li. E-mem: Multi-agent based episodic context reconstruction for llm agent memory. *arXiv preprint arXiv:2601.21714*, 2026.

[349] L. Wang, H. Chen, N. Yang, X. Huang, Z. Dou, and F. Wei. Chain-of-retrieval augmented generation. In *Advances in Neural Information Processing Systems*, 2025.

[350] L. Wang, L. Yang, B. Chen, K. Xu, G. Zou, B. Tang, F. Xiong, S. Chen, and Z. Li. Text2Mem: A unified memory operation language for memory operating system. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 2105–2119. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.100.

[351] R. Wang, Y. Shi, Z. Li, Z. Li, Y. Yu, J. Yang, K. Panaganti, H. Mi, D. Zhou, and Leoweiliang. Harness handbook: Making evolving agent harnesses readable, navigable, and editable. *arXiv preprint arXiv:2607.13285*, 2026.

[352] W. Wang, S. Li, T. Dong, Y. Meng, and H. Zhu. From function calls to MCPs for securing AI agent systems: Architecture, challenges and countermeasures. *ZTE Communications*, 23(3):27–37, 2025. doi: 10.12142/ZTECOM.202503004.

[353] W. Wang, P. Kattakinda, and S. Feizi. Do agent optimizers compound? a continual-learning evaluation on terminal-bench 2.0. *arXiv preprint arXiv:2607.14004*, 2026.

[354] X. Wang, C. Li, Z. Wang, F. Bai, H. Luo, J. Zhang, N. Jojic, E. P. Xing, and Z. Hu. PromptAgent: Strategic planning with language models enables expert-level prompt optimization. *arXiv preprint arXiv:2310.16427*, 2023.

[355] X. Wang, J. Wei, D. Schuurmans, Q. V. Le, E. H. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. In *International Conference on Learning Representations*, 2023.

[356] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji. Executable code actions elicit better LLM agents. *arXiv preprint arXiv:2402.01030*, 2024.

[357] X. Wang, Y. Chen, L. Yuan, Y. Zhang, Y. Li, H. Peng, and H. Ji. Executable code actions elicit better LLM agents. *arXiv preprint arXiv:2402.01030*, 2024. doi: 10.48550/arXiv.2402.01030. URL https://arxiv.org/abs/2402.01030.

[358] X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, Y. Song, B. Li, J. Singh, H. H. Tran, F. Li, R. Ma, M. Zheng, B. Qian, Y. Shao, N. Muennighoff, Y. Zhang, B. Hui, J. Lin, R. Brennan, H. Peng, H. Ji, and G. Neubig. OpenHands: An open platform for AI software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

[359] X. Wang, B. Li, Y. Song, F. F. Xu, X. Tang, M. Zhuge, J. Pan, Y. Song, B. Li, J. Singh, H. H. Tran, F. Li, R. Ma, M. Zheng, B. Qian, Y. Shao, N. Muennighoff, Y. Zhang, B. Hui, J. Lin, R. Brennan, H. Peng, H. Ji, and G. Neubig. OpenHands: An open platform for ai software developers as generalist agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=OJd3ayDDoF.

[360] Y. Wang, X. Ma, G. Zhang, Y. Ni, A. Chandra, S. Guo, W. Ren, A. Arulraj, X. He, Z. Jiang, T. Li, M. Ku, K. Wang, A. Zhuang, R. Fan, X. Yue, and W. Chen. MMLU-Pro: A more robust and challenging multi-task language understanding benchmark. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2406.01574.

[361] Y. Wang, S. Guo, Y. Pan, Z. Su, F. Chen, T. H. Luan, P. Li, J. Kang, and D. Niyato. Internet of agents: Fundamentals, applications, and challenges. *IEEE Transactions on Cognitive Communications and Networking*, 2025. doi: 10.1109/TCCN.2025.3623369. URL https://arxiv.org/abs/2505.07176.

[362] Y. Wang, Z. Wu, J. Yao, and J. Su. TDAG: A multi-agent framework based on dynamic task decomposition and agent generation. *Neural Networks*, 2025. URL https://arxiv.org/abs/2402.10178.

[363] Y. Wang, Z. Xu, Y. Huang, X. Wang, Z. Song, L. Gao, C. Wang, X. Tang, Y. Zhao, A. Cohan, X. Zhang, and X. Chen. DyFlow: Dynamic workflow framework for agentic reasoning. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2509.26062.

[364] Y. Wang, Y. Y. Gong, J. Li, Z. Zhu, and J. Li. Agentic information architectures for global climate governance: A multi-agent decision-support system for cross-national policy analytics. *Journal of Global Information Management (JGIM)*, 34(1):1–31, 2026.

[365] Y. Wang, X. Wang, Y. Yao, X. Li, X. Yang, Y. Teng, X. Ma, and Y. Wang. Agenticeval: Toward agentic and self-evolving safety evaluation of large language models. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 14789–14808, 2026.

[366] Y. Wang, Z. Zhang, M. Chi, K. Yu, Y. Li, M. Peng, B. Tong, C. Zhang, Y. Zhou, and J. Li. Evomembench: Benchmarking agent memory from a self-evolving perspective. *arXiv preprint arXiv:2605.18421*, 2026. URL https://arxiv.org/abs/2605.18421.

[367] Z. Wang, K. Wang, Q. Wang, P. Zhang, L. Li, Z. Yang, K. Yu, M. N. Nguyen, L. Liu, E. Gottlieb, et al. Ragen: Understanding self-evolution in llm agents via multi-turn reinforcement learning. *arXiv preprint arXiv:2504.20073*, 2025.

[368] Z. Wang, Y. Wang, X. Liu, L. Ding, M. Zhang, J. Liu, and M. Zhang. Agentdropout: Dynamic agent elimination for token-efficient and high-performance llm-based multi-agent collaboration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24013–24035, 2025.

[369] Z. Z. Wang, J. Mao, D. Fried, and G. Neubig. Agent workflow memory. *arXiv preprint arXiv:2409.07429*, 2024.

[370] J. Wei, M. Bosma, V. Y. Zhao, K. Guu, A. W. Yu, B. Lester, N. Du, A. M. Dai, and Q. V. Le. Finetuned language models are zero-shot learners. In *International Conference on Learning Representations*, 2022.

[371] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. V. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837, 2022.

[372] T. Wei, Z. Shi, M. Lin, B. He, Z. Liu, Y. Sang, Y. Bei, X. Ning, J. Zou, T.-W. Li, X. Lin, Y. Zhao, C. Wang, B. Dumoulin, D. Wang, J. He, and H. Lu. Evo-harness: Context-to-harness skill compilation for self-evolving agents. *arXiv preprint arXiv:2608.15071*, 2026.

[373] Y. Wei, Z. Huang, H. Li, W. W. Xing, T.-J. Lin, and L. He. VFlow: Discovering optimal agentic workflows for verilog generation. *arXiv preprint arXiv:2504.03723*, 2025. URL https://arxiv.org/abs/2504.03723.

[374] Z. Wei, W. Yao, Y. Liu, W. Zhang, Q. Lu, L. Qiu, C. Yu, P. Xu, C. Zhang, B. Yin, H. Yun, and L. Li. WebAgent-R1: Training web agents via end-to-end multi-turn reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 7920–7939. Association for Computational Linguistics, 2025.

[375] C. White, S. Dooley, M. Roberts, A. Pal, B. Feuer, S. Jain, R. Shwartz-Ziv, N. Jain, K. Saifullah, S. Dey, S. Agrawal, S. S. Sandha, S. V. Naidu, C. Hegde, Y. LeCun, T. Goldstein, W. Neiswanger, and M. Goldblum. LiveBench: A challenging, contamination-limited LLM benchmark. In *International Conference on Learning Representations*, 2025. URL https://arxiv.org/abs/2406.19314.

[376] N. Wiratunga, V. A. Wijayasekara, I. Nkisi-Orji, P. Salimi, K. Martin, and C. Bolaños. icare: Ontology-guided intent routing for multi-agent llm-based dialogue systems. *context (including current conversation goals)*, 5:6, 2025.

[377] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. Le Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45. Association for Computational Linguistics, 2020. URL https://aclanthology.org/2020.emnlp-demos.6/.

[378] G. Wölflein, D. Ferber, D. Truhn, O. Arandjelović, and J. N. Kather. Llm agents making agent tools. *arXiv preprint arXiv:2502.11705*, 2025.

[379] C. Wu, Z. Xiang, Y. Tang, Z. Chen, Q. Zhang, and J. Su. Memgraphrag: Memory-based multi-agent system for graph retrieval-augmented generation. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pages 5407–5418, 2026.

[380] D. Wu, H. Wang, W. Yu, Y. Zhang, K.-W. Chang, and D. Yu. Longmemeval: Benchmarking chat assistants on long-term interactive memory. In *International Conference on Learning Representations*, 2025.

[381] Q. Wu, G. Bansal, J. Zhang, Y. Wu, B. Li, E. Zhu, L. Jiang, X. Zhang, S. Zhang, J. Liu, A. H. Awadallah, R. W. White, D. Burger, and C. Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversation. In *Conference on Language Modeling*, 2024. URL https://arxiv.org/abs/2308.08155.

[382] T. Wu, Y. Li, Z. Tang, C. Jiang, L. Luo, G. Qi, S. Pan, and G. Haffari. Card: Towards conditional design of multi-agent topological structures. *arXiv preprint arXiv:2603.01089*, 2026.

[383] Y. Wu, T. Yue, S. Zhang, C. Wang, and Q. Wu. StateFlow: Enhancing LLM task-solving through state-driven workflows. *arXiv preprint arXiv:2403.11322*, 2024. doi: 10.48550/arXiv.2403.11322. URL https://arxiv.org/abs/2403.11322.

[384] Z. Wu, H. Zhang, F. Lin, W. Xu, X. Xu, Y. Chen, H. P. Zou, S. Chen, W. Zhang, X. Liu, P. S. Yu, and H. Wang. GAM: Hierarchical graph-based agentic memory for LLM agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 34647–34664. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.1600.

[385] Z. Xi, D. Yang, J. Liu, J. Huang, H. Guo, B. Huang, T. Chen, Q. Zhang, Z. Lu, C. Liu, J. Sun, J. Zhang, D. Zhu, X. Guo, J. Wang, Z. Zhang, Y. Yang, J. Ye, M. Gao, D. Liu, J. Ji, G. Li, T. Gui, Q. Zhang, and X. Huang. AgentGym2: Benchmarking

large language model agents in de-idealized real-world environments. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pages 44451–44479. Association for Computational Linguistics, 2026.

[386] Y. Xia and T. Wang. Researchloop: An evidence-gated control plane for ai-assisted research, 2026. URL https://arxiv.org/abs/2605.28282.

[387] Z. Xiang, Z. Chen, Y. Tang, Z. Wei, R. Ning, Y. Lin, Q. Zhang, and J. Su. MemSyco-Bench: Benchmarking sycophancy in agent memory. *arXiv preprint arXiv:2607.01071*, 2026.

[388] Z. Xiang, C. Wu, Q. Zhang, S. Chen, Z. Hong, X. Huang, and J. Su. When to use graphs in RAG: A comprehensive analysis for graph retrieval-augmented generation. In *International Conference on Learning Representations*, 2026.

[389] Z. Xiang, C. Yang, Z. Chen, Z. Wei, Y. Tang, Z. Teng, Z. Peng, Z. Li, C. Huang, Y. He, et al. A systematic survey of self-evolving agents: From model-centric to environment-driven co-evolution. 2026.

[390] L. Xiao, Z. Pan, Z. Wang, Z. Cao, and W. Li. Srefiner: Soft-braid attention for multi-agent trajectory refinement. In *2025 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 960–969. IEEE, 2025.

[391] R. Xiao, W. Ma, K. Wang, Y. Wu, J. Zhao, H. Wang, F. Huang, and Y. Li. Flowbench: Revisiting and benchmarking workflow-guided planning for llm-based agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 10883–10900, 2024. doi: 10.18653/v1/2024.findings-emnlp.638. URL https://aclanthology.org/2024.findings-emnlp.638/.

[392] Y. Xiao, C. Zhou, Y. Zhang, Q. Zhang, S. Dong, S. Chen, C. Yang, and X. Huang. Lag: Logic-augmented generation from a cartesian perspective. *arXiv preprint arXiv:2508.05509*, 2025.

[393] Y. Xiao, J. Chen, Q. Zhang, Y. Zhang, C. Zhou, L. Yang, L. Ren, X. Yang, and X. Huang. Logicpoison: Logical attacks on graph retrieval-augmented generation. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5575–5591, 2026.

[394] T. Xie, D. Zhang, J. Chen, X. Li, S. Zhao, R. Cao, T. J. Hua, Z. Cheng, D. Shin, F. Lei, Y. Liu, Y. Xu, S. Zhou, S. Savarese, C. Xiong, V. Zhong, and T. Yu. OSWorld: Benchmarking multimodal agents for open-ended tasks in real computer environments. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2024. URL https://arxiv.org/abs/2404.07972.

[395] A. Xin, J. Siow, J. Wang, Z. Yao, F. Zhang, J. Song, L. Hou, and J. Li. EurekAgent: Agent environment engineering is all you need for autonomous scientific discovery. *arXiv preprint arXiv:2606.13662*, 2026. doi: 10.48550/arXiv.2606.13662. URL https://arxiv.org/abs/2606.13662.

[396] Y. Xiong, J. Wang, B. Li, Y. Zhu, and Y. Zhao. Self-organizing agent network for LLM-based workflow automation. *arXiv preprint arXiv:2508.13732*, 2025. doi: 10.48550/arXiv.2508.13732. URL https://arxiv.org/abs/2508.13732.

[397] Z. Xiong, Y. Lin, W. Xie, P. He, Z. Liu, J. Tang, H. Lakkaraju, and Z. Xiang. How memory management impacts LLM agents: An empirical study of experience-following behavior. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 623–645. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.27.

[398] B. Xu, Z. Peng, B. Lei, S. Mukherjee, Y. Liu, and D. Xu. ReWOO: Decoupling reasoning from observations for efficient augmented language models. *arXiv preprint arXiv:2305.18323*, 2023. URL https://arxiv.org/abs/2305.18323.

[399] B. Xu, Y. Chen, J. Fang, R. Zhong, Y. Yao, Y. Zhu, L. Du, and S. Deng. Structmem: Structured memory for long-horizon behavior in llms. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 122–146, 2026.

[400] C. Xu, Y. Hu, R. Wang, X. Lin, W. Wang, D. Liu, and F. Feng. Tacomas: Test-time co-evolution of topology and capability in llm-based multi-agent systems. *arXiv preprint arXiv:2605.09539*, 2026.

[401] F. Xu, W. Shi, and E. Choi. RECOMP: Improving retrieval-augmented LMs with compression and selective augmentation. In *International Conference on Learning Representations*, 2024.

[402] F. F. Xu, Y. Song, B. Li, Y. Tang, K. Jain, M. Bao, Z. Z. Wang, X. Zhou, Z. Guo, et al. Theagentcompany: Benchmarking llm agents on consequential real world tasks. In *Advances in Neural Information Processing Systems, Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=LZnKNApvhG.

[403] H. Xu, X. Huang, Y. Liu, and Z. Deng. Tps-bench: Evaluating ai agents' tool planning and scheduling abilities in compounding tasks. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pages 34949–34961, 2026. doi: 10.18653/v1/2026.acl-long.1614. URL https://aclanthology.org/2026.acl-long.1614/.

[404] K. Xu, X. Lu, S. Qiao, Z. Ding, H. Xu, L. Liang, and N. Zhang. Longds-bench: On the failure of long-horizon agentic data analysis. *arXiv preprint arXiv:2605.30434*, 2026. URL https://arxiv.org/abs/2605.30434.

[405] W. Xu, Z. Liang, K. Mei, H. Gao, J. Tan, and Y. Zhang. A-MEM: Agentic memory for LLM agents. *arXiv preprint arXiv:2502.12110*, 2025.

[406] Y. Xu, W. Zhang, Y. Chen, X. Lin, and Y. Zhang. Self-evolving agents as dynamic graph transformation: A survey and new perspective, 2026.

[407] Z. Xu, N. Martelaro, and C. McComb. Supervising ralph wiggum: Exploring a metacognitive co-regulation agentic ai loop for engineering design. *arXiv preprint arXiv:2603.24768*, 2026.

[408] X. Xue, Z. Lu, D. Huang, Z. Wang, W. Ouyang, and L. Bai. Comfybench: Benchmarking llm-based agents in comfyui for autonomously designing collaborative ai systems. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025. URL https://arxiv.org/abs/2409.01392.

[409] D. Yan, J. Liang, D. Hu, R. He, N. J. Yuan, Q. Zhang, and T. Tan. Agentstream: How well do self-evolving llm agents perform under streaming tasks? *arXiv preprint arXiv:2608.00155*, 2026.

[410] A. Yang, A. Li, B. Yang, B. Zhang, B. Hui, B. Zheng, B. Yu, C. Gao, C. Huang, C. Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

[411] C. Yang, X. Wang, Y. Lu, H. Liu, Q. V. Le, D. Zhou, and X. Chen. Large language models as optimizers. In *International Conference on Learning Representations*, 2024.

[412] C. Yang, R. Wang, J. Jiang, Q. Jiang, Q. Zhang, Y. Deng, S. Li, S. Hu, B. Li, F. T. Pokorny, X. Huang, and X. Wang. Nondeterministic polynomial-time problem challenge: An ever-scaling reasoning benchmark for LLMs. *Transactions on Machine Learning Research*, 2026.

[413] C. Yang, Z. Xiang, Y. Tang, Z. Teng, C. Huang, F. Long, Y. Liu, and J. Su. Ttcs: Test-time curriculum synthesis for self-evolving. *arXiv preprint arXiv:2601.22628*, 2026.

[414] C. Yang, C. Zhou, Y. Xiao, S. Dong, L. Zhuang, Y. Zhang, Z. Wang, Z. Hong, Z. Yuan, Z. Xiang, S. Chen, H. Zhou, Q. Zhang, N. Liu, J. Su, X. Wang, Y. Chang, and X. Huang. Graph-based agent memory: Taxonomy, techniques, and applications. *arXiv preprint arXiv:2602.05665*, 2026. URL https://arxiv.org/abs/2602.05665.

[415] C. Yang, C. Zhou, Y. Xiao, S. Dong, L. Zhuang, Y. Zhang, Z. Wang, Z. Hong, Z. Yuan, Z. Xiang, et al. Graph-based agent memory: Taxonomy, techniques, and applications. *arXiv preprint arXiv:2602.05665*, 2026.

[416] D. Yang, A. Simoulin, X. Qian, X. Liu, Y. Cao, Z. Teng, and G. Yang. Docagent: A multi-agent system for automated code documentation generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 460–471, 2025.

[417] H. Yang, L. Lin, Y. She, X. Liao, J. Wang, R. Zhang, Y. Mo, and C. D. Wang. FinRobot: Generative business process ai agents for enterprise resource planning in finance. *arXiv preprint arXiv:2506.01423*, 2025. doi: 10.48550/arXiv.2506.01423. URL https://arxiv.org/abs/2506.01423.

[418] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press. SWE-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

[419] K. Yang, P. Li, Z. Wu, K. Xu, H. Huang, and X. Huang. Dart: Semantic recoverability for structured tool agents. *arXiv preprint arXiv:2605.23311*, 2026.

[420] W. Yang, D. Cao, J. Pang, M. Weng, and Y. Liu. Adaptive collaboration with humans: Metacognitive policy optimization for multi-agent llms with continual learning. *arXiv preprint arXiv:2603.07972*, 2026.

[421] X. Yang, J. Wang, B. Tang, X. Cheng, C. Liu, K. Zeng, and W. Jiang. When 20 agents fail to sort: The distributed sorting benchmark for scalable multi-agent systems. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026. doi: 10.18653/v1/2026.findings-acl.1698. URL https://aclanthology.org/2026.findings-acl.1698/.

[422] Y. Yang, Y. Zhang, M. Wu, K. Zhang, Y. Zhang, H. Yu, Y. Hu, and B. Wang. TwinMarket: A scalable behavioral and social simulation for financial markets. In *Advances in Neural Information Processing Sys-*

tems, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/5bf234ecf83cd77bc5b77a24ba9338b0-Abstract-Conference.html.

[423] Y. Yang, H. Chai, S. Shao, Y. Song, S. Qi, R. Rui, and W. Zhang. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems. *Advances in Neural Information Processing Systems*, 38:107309–107336, 2026.

[424] Y. Yang, H. Chai, S. Shao, Y. Song, S. Qi, R. Rui, and W. Zhang. Agentnet: Decentralized evolutionary coordination for llm-based multi-agent systems. *Advances in Neural Information Processing Systems*, 38:107309–107336, 2026.

[425] Y. Yang, Z. Gong, W. Huang, Q. Yang, Z. Zhou, Z. Huang, Y. Li, X. Gao, Q. Dai, B. Liu, K. Qiu, Y. Yang, D. Chen, X.-T. Yang, and C. Luo. SkillOpt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

[426] Z. Yang, W. Zeng, S. Jin, C. Qian, P. Luo, and W. Liu. Nader: Neural architecture design via multi-agent collaboration. In *2025 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4452–4461. IEEE, 2025.

[427] S. Yao, D. Yu, J. Zhao, I. Shafran, T. L. Griffiths, Y. Cao, and K. Narasimhan. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, 2023.

[428] S. Yao, J. Zhao, D. Yu, N. Du, I. Shafran, K. Narasimhan, and Y. Cao. ReAct: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

[429] S. Yao, N. Shinn, P. Razavi, and K. Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*, 2024. URL https://arxiv.org/abs/2406.12045.

[430] Y. Yao, X. Tan, C.-H. Liu, Y. Li, Z. Wang, W. Yu, Z. Tan, Y. Tian, G. Zhao, L. Sun, X. Zhang, and T. Yang. Harness-Bench: Measuring harness effects across models in realistic agent workflows. *arXiv preprint arXiv:2605.27922*, 2026.

[431] L. Yi, R. Lei, L. Yao, Y. Xie, Y. Li, W. Zhang, Z. Wei, Y. Li, and J.-Y. Nie. Learning agent-compatible context management for long-horizon tasks. *arXiv preprint arXiv:2605.30785*, 2026.

[432] C. Yu, Z. Cheng, H. Cui, Y. Gao, Z. Luo, Y. Wang, H. Zheng, and Y. Zhao. A survey on agent workflow—status and future. In *Proceedings of the IEEE International Conference on Artificial Intelligence and Big Data (ICAIBD)*, pages 770–781, 2025. doi: 10.1109/ICAIBD64986.2025.11082076. URL https://arxiv.org/abs/2508.01186.

[433] H. Yu, Z. Hong, Z. Cheng, K. Zhu, K. Xuan, J. Yao, T. Feng, and J. You. Researchtown: Simulator of human research community. *arXiv preprint arXiv:2412.17767*, 2024.

[434] H. Yu, T. Chen, J. Feng, J. Chen, W. Dai, Q. Yu, Y.-Q. Zhang, W.-Y. Ma, J. Liu, M. Wang, et al. Memagent: Reshaping long-context llm with multi-conv rl-based memory agent. In *International Conference on Learning Representations*, volume 2026, pages 39458–39486, 2026.

[435] J. Yu, Y. Ding, and H. Sato. Dyntaskmas: A dynamic task graph-driven framework for asynchronous and parallel llm-based multi-agent systems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 35, pages 288–296, 2025.

[436] Q. Yu, Z. Zhang, R. Zhu, Y. Yuan, X. Zuo, Y. Yue, T. Fan, G. Liu, L. Liu, X. Liu, H. Lin, Z. Lin, B. Ma, G. Sheng, Y. Tong, C. Zhang, et al. DAPO: An open-source LLM reinforcement learning system at scale. *arXiv preprint arXiv:2503.14476*, 2025.

[437] S. Yu, D. Chong, A. Nandi, D. Soylu, J. Sun, C. D. Manning, and W. Shi. Shepherd: Enabling programmable meta-agents via reversible agentic execution traces. *arXiv preprint arXiv:2605.10913*, 2026.

[438] Y. Yu, W. Ping, Z. Liu, B. Wang, J. You, C. Zhang, M. Shoeybi, and B. Catanzaro. RankRAG: Unifying context ranking with retrieval-augmented generation in LLMs. In *Advances in Neural Information Processing Systems*, volume 37, pages 121156–121184, 2024.

[439] Y. Yu, L. Yao, Y. Xie, Q. Tan, J. Feng, Y. Li, and L. Wu. Agentic memory: Learning unified long-term and short-term memory management for large language model agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21457–21483. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.981.

[440] D. Yuan, F. Lyu, Y. Yuan, W. Zhang, B. He, J. Geng, L. Du, Z. Sun, Y. Chen, C. Han, J. Kang, X. Chen, H. Wu, and X. Liu. Beyond message passing: A semantic view of agent communication protocols. *arXiv preprint arXiv:2604.02369*, 2026. doi: 10.48550/arXiv.2604.02369. URL https://arxiv.org/abs/2604.02369.

[441] L. Yuan, Y. Chen, X. Wang, Y. Fung, H. Peng, and H. Ji. CRAFT: Customizing llms by creating and retrieving from specialized toolsets. In *International Conference on Learning Representations*, 2024.

[442] S. Yuan, K. Song, J. Chen, X. Tan, D. Li, and D. Yang. Evoagent: Towards automatic multi-agent generation via evolutionary algorithms. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 6192–6217, 2025.

[443] L. Yue, K. R. Bhandari, C.-Y. Ko, D. Patel, S. Lin, N. Zhou, J. Gao, P.-Y. Chen, and S. Pan. From static templates to dynamic runtime graphs: A survey of workflow optimization for llm agents. *arXiv preprint arXiv:2603.22386*, 2026.

[444] X. Yue, Y. Ni, K. Zhang, T. Zheng, R. Liu, G. Zhang, S. Stevens, D. Jiang, W. Ren, Y. Sun, C. Wei, B. Yu, R. Yuan, R. Sun, M. Yin, B. Zheng, Z. Yang, Y. Liu, W. Huang, H. Sun, Y. Su, and W. Chen. MMMU: A massive multi-discipline multimodal understanding and reasoning benchmark for expert AGI. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9556–9567, 2024. URL https://openaccess.thecvf.com/content/CVPR2024/html/Yue_MMMU_A_Massive_Multi-discipline_Multimodal_Understanding_and_Reasoning_Benchmark_for_CVPR_2024_paper.html.

[445] Y. Yue, G. Zhang, B. Liu, G. Wan, K. Wang, D. Cheng, and Y. Qi. Masrouter: Learning to route llms for multi-agent systems. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15549–15572, 2025.

[446] M. Yuksekgonul, F. Bianchi, J. Boen, S. Liu, Z. Huang, C. Guestrin, and J. Zou. TextGrad: Automatic "differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

[447] S. Yun, J. Peng, P. Li, W. Fan, J. Chen, J. Y. Zou, G. Li, and T. Chen. Graph-of-agents: A graph-based framework for multi-agent llm collaboration. In *International Conference on Learning Representations*, volume 2026, pages 19745–19760, 2026.

[448] D. Zhang, Z. Li, X. Luo, X. Liu, P. Li, and W. Xu. MCP Security Bench (MSB): Benchmarking attacks against model context protocol in LLM agents. *arXiv preprint arXiv:2510.15994*, 2025.

[449] G. Zhang, Y. Yue, X. Sun, G. Wan, M. Yu, J. Fang, K. Wang, T. Chen, and D. Cheng. G-designer: Architecting multi-agent communication topologies via graph neural networks. *arXiv preprint arXiv:2410.11782*, 2024.

[450] G. Zhang, K. Chen, G. Wan, H. Chang, H. Cheng, K. Wang, S. Hu, and L. Bai. EvoFlow: Evolving diverse agentic workflows on the fly. *arXiv preprint arXiv:2502.07373*, 2025. URL https://arxiv.org/abs/2502.07373.

[451] G. Zhang, L. Niu, J. Fang, K. Wang, L. Bai, and X. Wang. Multi-agent architecture search via agentic supernet. *arXiv preprint arXiv:2502.04180*, 2025.

[452] G. Zhang, Y. Yue, Z. Li, S. Yun, G. Wan, K. Wang, D. Cheng, J. Yu, and T. Chen. Cut the crap: An economical communication pipeline for llm-based multi-agent systems. In *International Conference on Learning Representations*, volume 2025, pages 75389–75428, 2025.

[453] G. Zhang, M. Fu, and S. Yan. Memgen: Weaving generative latent memory for self-evolving agents. In *International Conference on Learning Representations*, volume 2026, pages 22555–22588, 2026.

[454] G. Zhang, H. Zhang, Y. Han, Y. Fan, Y. Shao, H. Tan, and R. Li. Learning to generate and extract: A multi-agent collaboration framework for zero-shot document-level event arguments extraction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pages 34665–34673, 2026.

[455] H. Zhang, S. Zhang, K. Li, C. Zhang, Y. Chen, Y. Zhang, L. Bai, and S. Hu. Self-harness: Harnesses that improve themselves. *arXiv preprint arXiv:2606.09498*, 2026.

[456] J. Zhang, J. Xiang, Z. Yu, F. Teng, X.-H. Chen, J. Chen, M. Zhuge, X. Cheng, S. Hong, J. Wang, B. Zheng, B. Liu, Y. Luo, and C. Wu. AFlow: Automating agentic workflow generation. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=z5uVAKwmjf.

[457] J. Zhang, Y. Yan, J. Yan, Z. Zheng, J. Piao, D. Jin, and Y. Li. A parallelized framework for simulating large-scale LLM agents with realistic environments and interactions. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)*, pages 1339–1349. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-industry.94. URL https://aclanthology.org/2025.acl-industry.94/.

[458] J. Zhang, C. Zhang, S. Chen, Z. Huang, P. Zheng, Z. Wang, P. Guo, F. Mo, S.-H. Bae, J. Zou, J. Wei, and Y. Yang. Lightweight LLM agent memory with small language models. In *Proceedings of the 64th Annual Meeting of the Association for Com-*

putational Linguistics (Volume 1: Long Papers), pages 12914–12929. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.588.

[459] M. Zhang, J. Kim, S. Xiang, J. Gao, and C. Cao. Dynamic role assignment for multi-agent debate. *arXiv preprint arXiv:2601.17152*, 2026.

[460] Q. Zhang, S. Chen, Y. Bei, Z. Yuan, H. Zhou, Z. Hong, H. Chen, Y. Xiao, C. Zhou, J. Dong, et al. A survey of graph retrieval-augmented generation for customized large language models. *arXiv preprint arXiv:2501.13958*, 2025.

[461] Q. Zhang, Z. Xiang, Y. Xiao, L. Wang, J. Li, X. Wang, and J. Su. Faithfulrag: Fact-level conflict modeling for context-faithful retrieval-augmented generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 21863–21882, 2025.

[462] Q. Zhang, Z. Feng, X. Shi, X. Hu, C. Liu, P. Xie, X. Wang, J. Ye, B. Hooi, H. Wang, and J. Zhao. SkillComposer: Learning to evolve agent skills for specification and generalization. *arXiv preprint arXiv:2606.06079*, 2026.

[463] Q. Zhang, C. Hu, S. Upasani, B. Ma, F. Hong, V. Kamanuru, J. Rainton, C. Wu, M. Ji, H. Li, U. Thakker, J. Zou, and K. Olukotun. Agentic context engineering: Evolving contexts for self-improving language models. In *International Conference on Learning Representations*, 2026. arXiv:2510.04618.

[464] S. Zhang, X. Ma, Z. Cao, Z. Zhang, and H. Zhao. Plan-over-graph: Towards parallelable LLM agent schedule. *arXiv preprint arXiv:2502.14563*, 2025. URL https://arxiv.org/abs/2502.14563.

[465] S. Zhang, M. Yin, J. Zhang, J. Liu, Z. Han, J. Zhang, B. Li, C. Wang, H. Wang, Y. Chen, and Q. Wu. Which agent causes task failures and when? on automated failure attribution of LLM multi-agent systems. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 76583–76599. PMLR, 2025.

[466] S. Zhang, C. Jiang, Z. Li, and J. Deng. Shapecraft: Llm agents for structured, textured and interactive 3d modeling. *Advances in Neural Information Processing Systems*, 38:65116–65144, 2026.

[467] S. Zhang, Y. Shi, and L. Wang. Patchboard: Schema-grounded state mutation for reliable and auditable LLM multi-agent collaboration. *arXiv preprint arXiv:2605.29313*, 2026.

[468] T. Zhang and Z. Qi. Skill-to-LoRA: From using skills to learning behaviors for token-efficient LLM agents. *arXiv preprint arXiv:2606.16769*, 2026.

[469] W. Zhang, X. Zhang, C. Zhang, L. Yang, J. Shang, Z. Wei, H. P. Zou, Z. Huang, Z. Wang, Y. Gao, et al. Personaagent: Bridging memory and action for personalized llm agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 26421–26439, 2026.

[470] X. Zhang, Z. Dou, D. Li, J. Tao, S. Cheng, R. Shi, F. Liu, E. Hu, Y. Ding, H. Wang, et al. Swarm skills: A portable, self-evolving multi-agent system specification for coordination engineering. *arXiv preprint arXiv:2605.10052*, 2026.

[471] Y. Zhang, C. Lin, S. Tang, H. Chen, S. Zhou, Y. Ma, and V. Tresp. Swarmagentic: Towards fully automated agentic system generation via swarm intelligence. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 1778–1818, 2025.

[472] Y. Zhang, F. Liu, Y. Shan, X. Huang, X. Yang, Y. Zhu, X. Cheng, C. Liu, K. Zeng, T. J. Zhang, and W. Jiang. Silo-bench: A scalable environment for evaluating distributed coordination in multi-agent llm systems. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics*, pages 29379–29398, 2026. doi: 10.18653/v1/2026.acl-long.1354. URL https://aclanthology.org/2026.acl-long.1354/.

[473] M. Zhao, X. Wei, Y. Shao, K. Zhou, L. Yang, S. Rao, J. Zhan, and Z. Chen. A2Flow: Automating agentic workflow generation via self-adaptive abstraction operators. *arXiv preprint arXiv:2511.20693*, 2025. URL https://arxiv.org/abs/2511.20693. Accepted to AAAI 2026.

[474] W. Zhao, C. Wu, Y. Fan, P. Qiu, X. Zhang, Y. Sun, X. Zhou, S. Zhang, Y. Peng, Y. Wang, X. Sun, Y. Zhang, Y. Yu, K. Sun, and W. Xie. An agentic system for rare disease diagnosis with traceable reasoning. *Nature*, 651:775–784, 2026. doi: 10.1038/s41586-025-10097-9. URL https://doi.org/10.1038/s41586-025-10097-9.

[475] X. Zhao, Z. Tan, V. Tadiparthi, N. Agarwal, K. Lee, E. M. Pari, H. N. Mahjoub, and T. Chen. Generative skill composition for llm agents. *arXiv preprint arXiv:2606.32025*, 2026.

[476] C. Zheng, J. Chen, Y. Lyu, W. Z. T. Ng, H. Zhang, Y.-S. Ong, I. Tsang, and H. Yin. MermaidFlow: Redefining agentic workflow generation via safety-constrained evolutionary programming. *arXiv preprint arXiv:2505.22967*, 2025. URL https://arxiv.org/abs/2505.22967.

[477] C. Zheng, S. Liu, M. Li, X.-H. Chen, B. Yu, C. Gao, K. Dang, Y. Liu, R. Men, A. Yang, J. Zhou, and J. Lin. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

[478] J. Zheng, X. Fang, J. Zhang, Z. Gui, H. Chen, and N. Zhang. Onedayagent: Towards a long-horizon harness for autonomous agents. *arXiv preprint arXiv:2608.05013*, 2026.

[479] L. Zheng, L. Yin, Z. Xie, C. Sun, J. Huang, C. H. Yu, S. Cao, C. Kozyrakis, I. Stoica, J. E. Gonzalez, C. Barrett, and Y. Sheng. SGLang: Efficient execution of structured language model programs. *arXiv preprint arXiv:2312.07104*, 2023. URL https://arxiv.org/abs/2312.07104.

[480] Y. Zheng, R. Zhang, J. Zhang, Y. Ye, and Z. Luo. LlamaFactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, pages 400–410. Association for Computational Linguistics, 2024. doi: 10.18653/v1/2024.acl-demos.38. URL https://aclanthology.org/2024.acl-demos.38/.

[481] H. Zhong and S. Zhu. AI harness engineering: A runtime substrate for foundation-model software agents. *arXiv preprint arXiv:2605.13357*, 2026.

[482] W. Zhong, L. Guo, Q. Gao, H. Ye, and Y. Wang. MemoryBank: Enhancing large language models with long-term memory. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 19724–19731, 2024.

[483] A. Zhou, K. Yan, M. Shlapentokh-Rothman, H. Wang, and Y.-X. Wang. Language agent tree search unifies reasoning, acting, and planning in language models. *arXiv preprint arXiv:2310.04406*, 2023.

[484] C. Zhou, H. Chai, W. Chen, Z. Guo, R. Shan, Y. Song, T. Xu, Y. Yang, A. Yu, W. Zhang, C. Zheng, J. Zhu, Z. Zheng, Z. Zhang, X. Lou, C. Zhang, Z. Fu, J. Wang, W. Liu, J. Lin, and W. Zhang. Externalization in LLM agents: A unified review of memory, skills, protocols and harness engineering. *arXiv preprint arXiv:2604.08224*, 2026.

[485] D. Zhou, N. Schärli, L. Hou, J. Wei, N. Scales, X. Wang, D. Schuurmans, C. Cui, O. Bousquet, Q. V. Le, and E. H. Chi. Least-to-most prompting enables complex reasoning in large language models. In *International Conference on Learning Representations*, 2023.

[486] H. Zhou, X. Wan, R. Sun, H. Palangi, S. Iqbal, I. Vulić, A. Korhonen, and S. Arik. Multi-agent design: Optimizing agents with better prompts and topologies. In *International Conference on Learning Representations*, volume 2026, pages 15844–15872, 2026.

[487] J. Zhou, T. Lu, S. Mishra, S. Brahma, S. Basu, Y. Luan, D. Zhou, and L. Hou. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*, 2023. URL https://arxiv.org/abs/2311.07911.

[488] S. Zhou, F. F. Xu, H. Zhu, X. Zhou, R. Lo, A. Sridhar, X. Cheng, T. Ou, Y. Bisk, D. Fried, U. Alon, and G. Neubig. WebArena: A realistic web environment for building autonomous agents. In *International Conference on Learning Representations*, 2024. URL https://arxiv.org/abs/2307.13854.

[489] S. Zhou, S. Yu, H. Wei, J. Wu, S. Ouyang, Y. Jiao, S. Pan, J. McAuley, Y. Zhang, T. Yu, and J. Han. Filesystem-based memory for LLM agents: Organization, evolution, and sustainability. *arXiv preprint arXiv:2607.26637*, 2026.

[490] X. Zhou, P. Bulter, C. Yang, S. D. Rihm, T. Angkanaporn, J. Akroyd, S. Mosbach, and M. Kraft. Ontology-to-tools compilation for executable semantic constraint enforcement in llm agents. *arXiv preprint arXiv:2602.03439*, 2026.

[491] Y. Zhou, A. I. Muresanu, Z. Han, K. Paster, S. Pitis, H. Chan, and J. Ba. Large language models are human-level prompt engineers. In *International Conference on Learning Representations*, 2023.

[492] Z. Zhou, A. Qu, Z. Wu, S. Kim, A. Prakash, D. Rus, B. K. H. Low, and P. Liang. Mem1: Learning to synergize memory and reasoning for efficient long-horizon agents. In *International Conference on Learning Representations*, volume 2026, pages 58413–58438, 2026.

[493] J. Zhu, J. Li, C. Zhang, J. Liu, and M. Yang. HeLa-Mem: Hebbian learning and associative memory for LLM agents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13757–13769. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.acl-long.625.

[494] K. Zhu, H. Du, Z. Hong, X. Yang, S. Guo, Z. Wang, Z. Wang, C. Qian, X. Tang, H. Ji, and J. You. Multiagentbench: Evaluating the collaboration and competition of llm agents. In *Proceedings of the 63rd Annual Meeting of*

the Association for Computational Linguistics, pages 8580–8622, 2025. doi: 10.18653/v1/2025.acl-long.421. URL https://aclanthology.org/2025.acl-long.421/.

[495] Y. Zhu, L. Liu, J. Yu, and D. Zhang. Llm-based multi-agent orchestration: A survey of frameworks, communication protocols, and emerging patterns. *Future Internet*, 18(6):326, 2026. doi: 10.3390/fi18060326.

[496] Z. Zhu, C. Xie, X. Lv, and slime Contributors. slime: An LLM post-training framework for RL scaling. GitHub repository, 2025. URL https://github.com/THUDM/slime.

[497] L. Zhuang, S. Chen, Y. Xiao, H. Zhou, Y. Zhang, H. Chen, Q. Zhang, and X. Huang. Linearrag: Linear graph retrieval augmented generation on large-scale corpora. In *International Conference on Learning Representations*, volume 2026, pages 147053–147075, 2026.

[498] M. Zhuge, W. Wang, L. Kirsch, F. Faccio, D. Khizbullin, and J. Schmidhuber. GPTSwarm: Language agents as optimizable graphs. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 62743–62767. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhuge24a.html.

[499] C. Zou, Y. Yao, S. She, and R. D. Hawkins. CalBench: Evaluating coordination-privacy trade-offs in multi-agent LLMs. *arXiv preprint arXiv:2605.09823*, 2026. URL https://arxiv.org/abs/2605.09823.

11 Appendix

11.1 Comparison with Related Surveys

Recent surveys examine LLM-based agents from several complementary perspectives. Broad agent surveys organize the field around planning and reasoning, memory, tool use, interaction with environments, and multi-agent coordination [170]. Graph–agent surveys focus more specifically on how graph structures enhance planning, execution, memory, tool use, reasoning, and multi-agent interaction [20, 218, 313]. These studies provide comprehensive accounts of agent capabilities and graph-enhanced agent functions, but their primary objective is to characterize or improve the capabilities of individual agents and multi-agent architectures rather than to organize the full engineering progression from models to system-level intelligence.

A second line of work focuses on the engineering infrastructure that turns foundation models into operational agents. Recent surveys trace the transition from prompting and context construction toward workflow and harness engineering, emphasizing that agent performance depends on both model capability and the surrounding execution infrastructure [107]. Harness-centered surveys further formalize execution loops, tool interfaces, context management, persistent state, lifecycle control, and evaluation as first-class runtime components [242]. Related studies examine externalized capabilities and code-centered harnesses [255, 484], while workflow surveys study planning, orchestration structures, executable workflows, and their optimization [432, 443]. These surveys substantially overlap with the transition from Model Intelligence to Individual Intelligence and with parts of task organization, but they generally take the individual agent runtime or executable workflow as their primary engineering object.

A third line of work moves toward system-level organization and evolution. Multi-agent orchestration surveys study task decomposition and allocation, coordination topology, communication protocols, state management, control-flow sequencing, failure recovery, and dynamic orchestration [495]. The LIFE survey connects individual capability, multi-agent collaboration, failure attribution, and autonomous self-evolution [284], while broader surveys of self-evolving agents organize how different agent components are persistently improved through experience and feedback [90]. Most closely related to our structural perspective, recent work formulates self-evolving agents as dynamic graph transformations, representing memories, tools, skills, workflows, and inter-agent relations as typed graph objects whose structures can evolve over time [406]. Its primary question is how agent evolution can be modeled and governed through dynamic graph transformation. Our survey instead begins from the organization of system intelligence and treats Task Organization, Agent Coordination, and Runtime State Management as explicit and interconnected system-level structures, with System Evolution describing how execution experience persistently improves these structures. Beyond structural organization and evolution, we further consider Ontology Engineering as a semantic foundation for defining shared system entities, relations, and constraints.

Table 4 compares representative surveys according to their substantive coverage of major topics in agent and system engineering. We focus on Harness, Loop, Planning, Workflow, multi-agent systems, Runtime State, Self-Evolution, and Ontology because these dimensions more directly distinguish how existing surveys treat the construction, execution, organization, and evolution of agent systems. Model-level topics such as foundation models, prompting, and context are not separately tabulated because they are widely used as general background across agent surveys and therefore provide limited discriminative value. Component-specific surveys devoted only to memory, tools, GraphRAG, evaluation, or individual application domains are not tabulated, although they remain important background references.

11.2 Distinction with Graph-based Approaches in Agents

The coverage comparison above clarifies which parts of agent and system engineering are addressed by existing surveys, but topic coverage alone does not capture the main distinction of Graph Engineering. The key difference lies in the architectural role assigned to graph structures. Existing graph–agent approaches typically use graphs to enhance particular capabilities, such as reasoning, planning, memory, retrieval, tool organization, workflow execution, or multi-agent communication. In these settings, the graph is primarily a representation or computational mechanism supporting an agent capability.

Graph Engineering instead treats explicit graph structures as the organizational substrate of the intelligent system. Task Organization represents goals, subtasks, dependencies, and executable workflows; Agent Coordination represents capabilities, responsibilities, team structures, and communication relations; and Runtime State Management represents

Table 4 : Topic coverage of representative surveys related to LLM agents and agent systems.

Survey / study	Harness	Loop	Planning	Workflow	MAS	State	Self-Evolution	Ontology
LLM Agents'26 [170]	○	–	✓	–	✓	–	–	–
Graphs Meet Agents'25 [20]	○	–	✓	○	✓	–	–	–
Graph-Aug. Agents'25 [218]	○	–	✓	○	✓	–	○	–
Agent Intelligence + Graphs'26 [313]	○	○	○	○	✓	–	–	–
QA-to-Task Completion'26 [107]	✓	✓	○	✓	○	✓	✓	–
Agent Harness'26 [242]	✓	✓	✓	○	✓	✓	○	–
Runtime Graphs'26 [443]	○	○	○	✓	○	○	○	–
Multi-Agent Orchestration'26 [495]	○	○	✓	✓	✓	✓	–	–
Beyond Individual'26 [284]	○	○	○	○	✓	○	✓	–
Dynamic Graph Transform.'26 [406]	○	–	○	✓	✓	✓	✓	○
Ours	✓	✓	✓	✓	✓	✓	✓	✓

✓: primary organizing axis or dedicated taxonomy; ○: substantive secondary coverage; –: absent, incidental, or only briefly mentioned as background. **Harness**: extra-model capabilities and infrastructure such as tools, memory, skills, interfaces, execution environments, or runtime governance; **Loop**: explicit iterative control through action, observation, feedback, verification, recovery, or termination; **Planning**: task decomposition, dependency structuring, scheduling, or allocation; **Workflow**: construction, execution, or optimization of multi-step computational workflows; **MAS**: multi-agent roles, allocation, team organization, communication, topology, or orchestration; **State**: persistent runtime state, provenance, consistency, failure localization, or recovery; **Self-Evolution**: experience-driven improvements that persist across executions; **Ontology**: explicit ontology engineering or shared machine-interpretable semantics for system entities, relations, and constraints. Symbols indicate substantive survey scope rather than paper quality; brief background mentions are not counted.

execution state, provenance, failures, and recovery dependencies. These structures are coupled: changes in task organization can alter capability requirements and agent allocation, changes in agent organization can affect communication and execution assumptions, and runtime evidence can trigger revisions to task and agent structures. System Evolution further turns execution experience into persistent structural improvements that can be validated, retained, reused, or rolled back across executions. Ontology Engineering complements these structures by providing shared, machine-interpretable definitions of system entities, relations, and constraints, thereby supplying a semantic foundation for their consistent interpretation and reuse.

The recent dynamic-graph view of self-evolving agents is particularly close to this perspective because it also treats agent components and relations as explicit graph objects that can change over time [406]. The distinction is primarily one of organizing question and system scope. Dynamic graph transformation begins from persistent agent evolution and asks how memories, tools, skills, workflows, and inter-agent relations can be represented and rewritten as evolving graph structures. Graph Engineering begins from the organization of System Intelligence and asks how task structures, acting entities, and runtime states should be jointly represented, coordinated, governed, and improved. Evolution is therefore one dimension of Graph Engineering rather than its sole organizing axis.

This system-level view also clarifies the relationship between Graph Engineering and earlier engineering paradigms. *Prompt Engineering* and *Context Engineering* determine how model capabilities are elicited and what information is available at inference time. *Harness Engineering* provides persistent and executable capabilities around the model, while *Loop Engineering* organizes these capabilities into bounded, feedback-driven, and goal-directed execution. Graph Engineering addresses the next organizational scale: how multiple tasks, agents, resources, and evolving runtime states should be explicitly structured and coordinated as a coherent system. Ontology Engineering further provides a shared semantic model through which these system structures can be consistently interpreted and connected. In this sense, the progression from Model Intelligence to Individual Intelligence and ultimately System Intelligence is not defined by adding more components, but by expanding the engineering object from model behavior, to persistent agent execution, and finally to the explicit organization, evolution, and semantic grounding of system-level relationships.